COSTRUISCI IL TUO FIREFOX

PERSONALIZZA FACILMENTE IL BROWSER PIÙ IN VOGA DEL MOMENTO CON IL LINGUAGGIO XUL

Rivista + Le Grandi Guide di ioProgrammo n° 1 o n° 2 a € 14,90 in più **RIVISTA+CD €6,90**

PER ESPERTI E PRINCIPIANTI

Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL Periodicità mensile • LUGLIO/AGOSTO 2005 • ANNO IX, N.7 (93)

ioProgrammo Anno IX - N°7 (93)

GESTIONE DELL'AUDIO DEL CELLULARE - UN PLAYER MP3 IN VISUAL BASIC - COSTRUIRE UN ROBOT PER IRC - INTELLIGENZA ARTIFICIALE

EDIZIONI MASTER

IL CELLULARE RISPONDE PER TE E...

REGISTRA LA VOCE

CREA UNA SEGRETERIA CHE FUNZIONA A TELEFONO ACCESO, NON TI FA PERDERE LE TELEFONATE E NON HA COSTI!

- Intercetta lo squillo e risponde in modo automatico
- Registra le voce di chi chiama in formato wav
- Ti fa ascoltare con il proprio player il messaggio registrato

VISUAL BASIC TE LE SUONA!

Sviluppa il tuo player MP3 o il tuo lettore DVD e dotalo di funzioni che gli altri non hanno

UN ROBOT PER IRC

Crealo in Java, programmalo, e fagli prendere il tuo posto nelle chat quando non ci sei

C++

INTERFACCE MULTIPIATTAFORMA

Programma applicazioni sia per Linux sia per Windows usando le WxWidgets

JAVA

INTELLIGENZA ARTIFICIALE

Emula il cervello umano con il PC istruendo una rete neurale

IOPROGRAMMO WEB

WEBSERVICE USALI CON PHP

Sfrutta servizi esistenti, aggiungili al tuo sito e impara a crearne di nuovi

PAGINAZIONE DA RECORD

Rendi veloce il tuo sito anche quando il numero di dati è elevato

VIDEOGAMING

PIXEL SHADER

Quando la modellazione non è un gioco, ecco come i personaggi diventano veri

C#

GARBAGE COLLECTION

Sfrutta al massimo la memoria per rendere la tua applicazione velocissima

VB.NET

OCR FAI DA TE

Scansiona un documento e riconosci il testo usando i componenti di Office

DATAGRID SU MISURA

Fai diventare le griglie dei contenitori di immagini e testo

DATABASE

PERSISTENZA DEI DATI

Sfrutta la semplicità di Castor per trasformare i DB in Classi Java

ALGORITMI

SFRUTTA IL SISTEMA

Concorrenza e semafori, come non intasare le risorse

INTERVISTA ESCLUSIVA: CHRIS ATWOOD RESPONSABILE DEL TEAM DI SUN JAVA STUDIO CI ILLUSTRA LE CARTE VINCENTI DELL'AMBIENTE

Anno IX - N.ro 7 (93) - Luglio/Agosto 2005 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Redazione: Raffaele del Monaco, Fabio Farnesi
Collaboratori: M. Autiero, D. Berta, M. Bigatti, D. Bochicchio, L. Buono, D. De Michelis, A. Galeazzi, F. Grimaldi, M. Ianigro, A. Marroccelli, L. Mattei , G. Negrelli, F. Paparoni, A. Pelleriti, M. Scala, F. Smelzo, A. Trapani, F. Vaccaro, S. Vena.
Segreteria di Redazione Veronica Longo

Realizzazione grafica: Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale: SET S.r.l.
Coordinamento Tecnico: Piero Mannelli
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Tel. 02 831213 - Fax 02 83121330
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 20% sul prezzo di copertina di €75,90
Offerte valide fino al 30/09/05
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel. 02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelliera, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Neotek S.r.l.- C.da Imperatore - zona ASI Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Giugno 2005

Edizioni Master edita: Computer Bild, Idea Web, Go!OnLine Internet Magazine, Win Magazine, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, Filmteca in DVD, La mia videoteca, TV e Satellite, Win Extra, Home entertainment, Digital Japan, Digital Music, Horror Mania, ioProgrammo Extra, Le Collection.

Programmazione Embedded

A molti non sfuggirà la presentazione della scheda Fox Micro Linux System che vi proponiamo nelle pagine di questo stesso numero. Non voglio parlarvene ancora nell'editoriale per sottolinearne il valore assoluto, che pure rimane alto, ma per puntare l'indice su quello che è il futuro della programmazione. Fino ad ora abbiamo visto un'evoluzione del software dalle interfacce visuali a quelle grafiche e infine al Web. È tempo di un ulteriore passo avanti, a farsi largo sono i sistemi embedded. Non parliamo solo di telefonini, palmari e pocket PC, ma di tutta una serie di periferiche che a vario titolo contengono un processore programmabile. Parliamo di programmazione di controlli per l'automazione industriale come di domotica, parliamo di automobili come di aeroplani, parliamo di robot come di applicazioni per il controllo dei dati. È un nuovo modo di programmare verso cui dobbiamo tendere, è un mercato verso il quale noi programmatori dobbiamo indirizzarci se vogliamo restare al passo con i tempi. Perciò iniziate a sviluppare piccole applicazioni non standalone e non dedicate al web, sicuramente in breve tempo sarete in grado di proporre ai vostri clienti tutta una serie di soluzioni che al momento sono impensabili. Noi di ioProgrammo non mancheremo, come sempre, di supportarvi lungo questa strada, mostrandovi come essa sia percorribile, così come abbiamo fatto quando era tempo di aiutarvi a sviluppare interfacce grafiche, così come abbiamo fatto quando vi abbiamo aiutato a programmare il Web, così come ancora faremo ogni volta che una nuova sfida si proporrà al mondo in cui tanto crediamo: quello della programmazione.

Fabio Farnesi

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *soft**codice*\ e *soft**tools*\) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **mare** *Password:* **barca**

pag. 14

IL PIXEL CHE TI CAMBIA I CONNOTATI

Alla scoperta del Pixel Shading, per realizzare animazioni vicine alla qualità cinematografica pag. 48

http://forum.ioprogrammo.it

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

RIVISTA + CD-ROM in edicola

IL SOFTWARE DEL MESE

Questo mese con ioProgrammo trovi Sybase PocketBuilder, la soluzione per lo sviluppo RAD di applicazioni dedicate al mondo Mobile

Si tratta di un prodotto innovativo, da provare per quanti intendono dedicarsi ai nuovi device,intesi come Pocket PC o Smartphone. Il nome del produttore: Sybase é una garanzia di affidabilità, e certifica anche un'alta integrazione con applicazioni di databse. Se pensate che il Mobile sia un settore emergente non potete fare a meno di provarlo.

I contenuti del CD-Rom

Apache 1.3.33/2.0.54
Il Web Server più usato al mondo
Directory: /Apache

Castor 0.9.6
Un tool di persistenza dei dati leggero
Directory: /Castor

Commons Collection 3.1
Aumenta la velocità di sviluppo delle applicazioni Java
Directory /CommonCollections

Commons HttpClient 3.0
Accedere al web via Java
Directory /CommonHttpClient

Commons Net 1.4.0
Internet lato client sotto controllo
Directory /CommonNet

Commons Oro 2.0.8
Espressioni regolari anche in Java
Directory /CommonsOro

Commons VFS
Supporto universale a qualunque tipo di File System
Directory /CommonsVFS

DbaMGR 2k
Il gestore di database MSDE
Directory /DBAmgr2K

Dev C++ 4.9.9.2
Il più amato dai programmatori C++
Directory /DevCPP

DevPHP
Programmazione PHP facilitata
Directory /DevPHP

Eclipse Sdk 3.0.2
L'IDE di programmazione universale
Directory /Eclipse

Encache 1.1
Una cache per le applicazioni Java
Directory /encache

Irrlicht 0.9
La libreria per lo sviluppo rapido di VideoGames
Directory /irrlicht

J2SE 1.5.0_03
L'sdk essenziale per programmare in Java
Directory /J2SE

Jakarta SLIDE WebDavCLient
Le librerie Java per il supporto a WebDav
Directory /jakartaslide

JSH 0.1.2.0
Una shell per Java.
Directory /JSH

Samba e CIFS in un un'unica libreria
Una libreria per accedere a Samba da Java
Directory /Jcifs

Lazarus 0.9.6
Quasi come Delphi ma Gratis
Directory /Lazarus

MySQL 4.11/5.0.4
Il principe dei database
Directory /MySQL

PHP 4.3.11 - 5.0.4
Il linguaggio di internet
Directory /PHP

Ultimate ++
L'IDE più innovativo per C++
Directory /Ultimatepp

Python 2.4.1.
Il linguaggio emergente
Directory /Python

SharpDevelop 1.0.3
L'ambiente alternativo per la programmazione C#
Directory /SharpDevelop

Pircbot
L'irc programmabile in Java
Directory /Pircbot

Thing 0.1
L'editor di Thinlet
Directory /Thing

Tomcat 5.5.9
L'application server per JSP
Directory /Tomcat

Jakarta Commons Logging
Le api essenziali per il Debugging
Directory /logging

Snarli
La libreria Java per lo sviluppo di reti neurali
Directory: /Snarli

Xerces 1.4.4
Il parser XML per Java e C+
Directory: /Xerces

Xalan
Il processore XSLT per Java e C++
Directory: /xalan

OpenRpt 1.1.1 beta Win32
Un sistema di reportistica completa ed OpenSource.
Directory: /openrpt

Dev C++ Pack
Tre package per estendere Dev C++
directory: /devpack

wxDevCpp
un Dev C++ potenziato
Directory: /wxWidgets

News

WINDOWS MEDIA CENTER NELL'XBOX 360

La prossima console di Microsoft, l'XBox 360, integrerà il sistema operativo Windows Media Center 2005. L'integrazione di questo strumento con la gia potente console di Microsoft rappresenta un ulteriore passo avanti nella convergenza dei mercati destinati all'Home Entertainment. In realtà una prima integrazione con il Windows Media Center è stata già tentata con la versione attuale, ma il passaggio alla nuova release del sistema dovrebbe garantire prestazioni più elevate e una maggiore interattività.

F-SECURE PORTA L'ANTIVIRUS SUI CELLULARI

La grande diffusione di sistemi mobili ha portato, parallelamente, alla diffusione di virus e sistemi di intrusione dedicati a cellulari e SmartPhone.
Fra i primi a scendere in campo con un prodotto dedicato alla protezione di questi dispositivi da attacchi esterni è stata la finlandese F-Secure. Il mobile antivirus si installa sugli smartphone proteggendoli da contenuti dannosi, inoltre è facilmente aggiornabile tramite un sistema automatico. F-Secure dispone così di una gamma completa di prodotti dedicati alla sicurezza per tutte le piattaforme, dai sistemi dedicati alle aziende, all'uso personale e ora anche sistemi per la protezione del mobile sia lato provider che utente finale, entrando così alla grande in un mercato tutto da scoprire.

DIECI ANNI DI JAVA

Si è svolta il 22 e il 23 Giugno 2005 a Milano la decima "Java Conference", evento annuale promosso da Sun e rivolto ai professionisti dell'IT e ai vertici aziendali che desiderano stare al passo con le tendenze di settore. I numeri di Java sono impressionanti:

quattro milioni e mezzo sono gli sviluppatori che utilizzano il linguaggio di Sun. Java è assolutamente il leader indiscusso del settore della telefonia mobile, dei palmari Java enabled e in generale di tutto il segmento dei sistemi embedded. Per quanto riguarda il Web la stima giornaliera di utilizzo di Java dai parte dei Surfer è di circa il 90%.
Questi numeri testimoniano un successo spesso poco gridato ma che nei fatti assume proporzioni notevoli. Il tema della Conference di quest'anno è stato l'OpenSource, segno evidente dell'attenzione che Sun pone nei confronti della disponibilità del codice sorgente, nella creazione degli standard e nella volontà di mettere in grado gli sviluppatori di poter cooperare fra loro.
La posizione di Sun rispetto all'OpenSource è chiara: si tratta del propulsore più efficiente per promuovere l'innovazione e lo sviluppo non solo da un punto di vista strettamente tecnologico ma anche da un punto di vista economico.

IL GOSSIP DI IIS7

Internet Information Server è il prodotto di Microsoft che aspira a conquistare posizioni di rilievo nel settore dei prodotti dedicati alla gestione di Applicazioni Web. Come già il suo rivale Apache, IIS si presenta come un prodotto completo che non contiene solo la logica di gestione delle pagine HTML, piuttosto si propone come container di applicazioni il cui sottostrato naturale è la rete. A differenza di Apache, IIS focalizza le sue caratteristiche su applicazioni basate sull'ormai arcinota piattaforma .NET.
I programmatori, soprattutto quelli orientati alla programmazione di Web Application sono rimasti freddi di fronte all'arrivo di .NET prima e di IIS6 suo naturale container dopo. Nonostante questo, Microsoft sta già lavorando alla versione 7 di *Internet Information Server,* il cui core sembra essere proprio una maggiore integrazione con Asp .NET. L'altra novità che sembra emergere dai pettegolezzi circolanti nei laboratori della società di Bill Gates è relativa ad una maggiore modularizzazione di IIS, che dovrebbe diminuire il rischio di attacchi esterni. Queste due novità nella loro semplicità potrebbero non rappresentare un salto sufficiente a giustificare un cambio di versione così marcato, ci sono dunque altre caratteristiche che al momento non trapelano dalle bocche cucite dei progettisti di casa Microsoft. Tuttavia alcune altre indiscrezioni sembrano indicare un cambio anche nel sistema di

EXCEL SOTTO ACCUSA

Il sig. Carlos Amado Amado, ex studente di Stanford, avrebbe sporto denuncia contro Micosoft per l'uso indebito di un brevetto da lui depositato nel 1990 per collegare Excel ad Access attraverso un unico foglio di calcolo. Secondo quanto riportato da Amado nel 1992 avrebbe tentato senza successo di vendere il brevetto a Microsoft, che invece se ne sarebbe appropriata indebitamente poco più tardi. Nel caso di accertata violazione, la valutazione del danno ammonterebbe a circa 2 euro per copia di Office venduta, per un totale di circa cinquecento milioni di dollari. Naturalmente Microsoft ha negato di avere in alcun modo utilizzato il brevetto di Amado, al contrario non si è per niente scomposta. Il legale del gigante di Redmond: Joel Freed ha dichiarato che Microsoft aveva iniziato a lavorare a questa tecnologia già tre anni prima, cioè nel 1989 in tempi assolutamente non sospetti. D'altra parte la società di Gates è abituata a questo genere di controversie legali. Con questa di Amado, salirebbero a 34 le contese a cui il gruppo sta facendo fronte in relazione a violazione di brevetti software.
A margine di questa vicenda c'é da sottolineare che in casi del genere é davvero difficoltoso stabilire una regola certa sulla base della quale emettere un giudizio.

configurazione di IIS7. Se fino a IIS6 eravamo abituati a fare i conti con una sorta di file registry piuttosto complesso, in IIS7 tutta la gestione sembra essere demandata a file XML ai quali però verrà affiancato un sistema di criptazione dei dati che li proteggerà da occhi indiscreti. L'idea comunque è quella di fornire agli sviluppatori dei metodi che gli consentano di settare le applicazioni secondo le loro esigenze senza per questo dovere passare attraverso l'amministratore del sistema.
Non si hanno ancora notizie certe circa l'arrivo sul mercato di questo nuovo prodotto della famiglia Microsoft, ma alcune altre informazioni potrebbero venire fuori per la fine dell'anno. Se da un lato IIS7 sicuramente presenterà delle novità importanti rispetto al suo predecessore, la vera battaglia si gioca sulla diffusione della piattaforma .NET. ovvero sulla volontà dei programmatori di compiere una migrazione che si sta rivelando più dolorosa di quanto non ci si aspettasse.

GOOGLE DIVENTA PERSONALIZZATO

È disponibile all'indirizzo *www.google.com/ig* il nuovo servizio di Google che consente di personalizzare l'home page con dei contenuti prelevati dai servizi più diffusi. Al momento è possibile visualizzare nella propria Home Page un'anteprima della Gmail se ne possedete una, le ultime notizie da news.google.com, oppure gli ultimi aggiornamenti di Wired oppure Slashdot. La parte economica/finanziaria e comunque più di cronaca è salvaguardata dalla presenza del New York Times e di BBC News, nel tempo probabilmente verranno aggiunti ulteriori servizi. E questo è l'ennesimo servizio che gli uomini del Marketing di Google prima e i loro sapienti programmatori dopo, mettono a disposizione del pubblico della rete. La continua crescita del leader fra i motori di ricerca è davvero notevole ed è forse l'unico colosso informatico in grado nel tempo di offrire una gamma di servizi completi come quelli offerti ad esempio da Microsoft.
I due partono da poli diversi, il primo trova le fondamenta in una massiccia presenza sulla rete, il secondo nello sviluppo del software, ambedue migrano i loro contenuti verso il polo di attrazione dell'altro. Quando avverrà lo scontro?

PRIME INDISCREZIONI SU OFFICE 12

Il Gossip di stanpo Microsoftiano ha iniziato a mormorare circa il rilascio di Office 12, l'attesa nuova versione della suite più diffusa al mondo. Sembrerebbe che il nuovo nato debba vedere la luce nell'estate del 2006, anche se nessuna conferma ufficiale è arrivata dallo staff di Microsoft. Tuttavia si mormora che secondo il calendario interno del team di sviluppo la prima beta dovrebbe essere disponibile per la fine di Agosto 2005, le successive beta release sono previste per Dicembre 2005 e Maggio 2006 per poi arrivare al rilascio definitivo per Luglio 2006. Nessuna dichiarazione rispetto a queste date è stata rilasciata da Microsoft, che si limita soltanto ad affermare che Office 12 segue un ciclo di vita diverso rispetto a Longhorn la cui uscita è prevista proprio per il 2006, inoltre la nuova versione della suite per l'ufficio di Microsoft sarà progettata per funzionare nel range di sistemi operativi che va da Windows 2000 fino a Windows server 2003 e superiori, anche se probabilmente un nuovo service pack dovrà essere rilasciato per consentire ai vecchi sistemi di far funzionare Office 12. Nessuna indiscrezione è fuoriuscita circa le nuove funzionalità di cui il prodotto sarà dotato. Alcune voci sembrerebbero indicare, però, che la novità principale sarà la presenza di un nuovo tool per la gestione dei diagrammi, che dovrebbe fare capo ad Avalon che a sua volta è parte di Longhorn. Questo implicherebbe una sterzata decisa di Office verso un target "manageriale" che userebbe il sistema come un prodotto integrato per l'intera gestione del ciclo di business aziendale.

HOTMAIL CORTEGGIA I POSTMASTER

È ancora una volta lo Spam sotto accusa. Tutti lo combattono, lo reputano a ragione un male oscuro che rallenta la rete internet intasandola senza ragione. MSN Hotmail, storico pioniere dei servizi di posta elettronica gratuiti ha implementato un tale numero di filtri e protezioni che adesso ha sentito il bisogno di creare un sito centrale che supporti i postmaster fornendogli informazioni su come sia possibile inviare posta agli utenti hotmail senza incorrere in problemi. Sul sito vengono spiegate le caratteristiche dei filtri utilizzati da MSN contro lo spam e i virus, e contemporaneamente vengono messi a disposizione degli ISP strumenti per il monitoring del traffico di posta da e verso MSN al fine di ottimizzare gli scambi con Hotmail.
Al di la di quanto utile realmente si dimostrerà questo servizio, è interessante notare come ormai proteggere troppo equivalga quasi a non proteggere per niente, di fatto l'istituzione di un servizio del genere indica che buona parte dei messaggi rivolti a utenti Hotmail generano dei falsi positivi e finiscono per essere filtrati anche quando non necessario.

Chiarezza sulle licenze

Buongiorno, mi chiamo Marco e sono da sempre un appassionato di tutto ciò che è tecnologia. A questa mia passione non poteva non corrispondere un interesse per la programmazione che è il motore e il "cervello" di tutti gli apparati meccanici/elettronici che fanno da spina dorsale alla tecnologia. Come appassionato del progresso ho sempre creduto nel valore della GPL che rendendo il sorgente di un software disponibili a tutti, mette in grado, chi ha le capacità, di contribuire in modo massiccio alla sua evoluzione. Recentemente però sono nate centinaia di licenze OpenSource o similari. Potreste fare un po' di chiarezza su tutte queste licenze?

Marco da Treviso

Gentile Marco, prima di parlare di licenze è importante fare una differenza fondamentale più sottile ma su cui ognuno dovrebbe riflettere per poi effettuare una scelta di campo, stiamo parlando della differenza fra il modello Free Software e il modello OpenSource. Spesso i due termini sono usati l'uno in sostituzione dell'altro, eppure le differenze sono tali da influire sulla concezione della società che ciascuno di noi dovrebbe formarsi. Tutte e due i modelli condividono i seguenti punti basati sulla licenza GPL

- Il software prodotto sotto licenza GPL può essere copiato e distribuito a condizione che ne sia reso disponibile il codice sorgente
- Chiunque può modificare il codice di un programma e ridistribuire il programma modificato purché il prodotto così modificato sia distribuito ancora una volta sotto licenza GPL, siano evidenti le modifiche apportate, siano mantenuti i riferimenti all'autore e al software originale.

La licenza GPL è sufficientemente più complessa, una copia integrale è disponibile all'indirizzo: *http://www.fsf.org/licensing/licenses/gpl.html*, tuttavia per sottolineare la differenza fra OpenSource e FreeSoftware i due punti in questione sono più che sufficienti. È evidente che rispettando la GPL si ha un effetto a catena che contrappone un modello di diffusione del software "libero" a un modello "proprietario" dove per ottenere il diritto d'uso del software è necessario corrispondere un compenso all'autore. Inoltre il software proprietario non può essere modificato in alcun modo, riprodotto e ridistribuito. Il movimento denominato Free Software utilizza la GPL per una questione di libertà sostenendo che il software aiuta l'umanità a progredire; pertanto, dovrebbe potere essere usato da tutti senza limitazioni e dovrebbe essere possibile modificarlo perché questo consente a chi ne ha le capacità di migliorarlo e contribuire così al progredire della qualità della vita. Per il Free Software ci sono delle basi filosofiche, per cui produrre applicazioni che adottino la GPL.
Queste basi corrispondono a un modello di società più libera e non basata sugli attuali vincoli di "business", ma su una visione che vede alla base la solidarietà tra le persone e la capacità di aiutarsi a vicenda. L'OpenSource viceversa sostiene l'utilizzo della GPL per un motivo opposto e cioè perché produrre software utilizzando questo modello è un buon modo di fare business. L'OpenSource sostiene che questo tipo di licenza può tranquillamente sostenere l'economia di un'azienda. Pertanto pur partendo dalle stesse basi, i due movimenti abbracciano ideali diversi.
Su questa onda si sono prodotte tantissime licenze, molte delle quali derivate dalla GPL. Alcune aderenti alla filosofia del Free Software, altre a quella dell'OpenSource. Una lista abbastanza completa di tutte le licenze è disponibile all'indirizzo *www.fsf.org/licensing/licenses/index_html#GNUGPL*. In questa pagina le licenze vengono divise in:

- licenze sul software;
- licenze sulla documentazione.

Queste due tipologie vengono a loro volta divise in

- licenze compatibili con la GPL.
- licenze incompatibili con la GPL ma comunque aderenti al Free Software.
- licenze incompatibili con Free Software.

In questo modo scopriamo ad esempio che la LGPL *(lesser general public licence)* consente di linkare al software dei moduli non GPL o che la licenza di Eclipse è incompatibile con la GPL perché ha al suo interno alcune limitazioni che non la rendono completamente "Libera". Riassumendo: al di là delle sigle e dei singoli punti di ciascuna licenza, è importante capire se si vuole aderire a un modello di società "libero" abbracciando la filosofia del "Free Software" oppure se si crede comunque in un modello di società per cui il "Business" ha una valenza importante. Nel secondo caso abbracciare un modello OpenSource o proprietario, significherà semplicemente scegliere un modello di marketing differente per i vostri prodotti, nel primo caso invece significherà investire in un modello di società differente, forse utopico ma sicuramente diverso. È utile dire a questo proposito, che se Richard Stal-

lmann non avesse creduto in questo modello di società investendo gran parte della sua vita in Free Software Foundation, forse oggi il mondo, almeno quello informatico, sarebbe meno alla portata di tutti e più controllato da pochi. Il che forse avrebbe reso qualche programmatore più ricco, ma il mondo un po' peggiore.

Informazioni su SPE

Mi sono cimentato da poco nell'apprendimento di Python che mi pare un ottimo linguaggio per software. Sono in possesso dell'editor SPE sia quello distribuito nel numero 90 della rivista che l'ultima versione scaricata dalla rete. Il problema è che una volta installato non riesco ad avviarlo: si creano diversi file che vengono avviati dal command line di Python (per altro senza successo) e non trovo un file .exe. A questo punto chiedo: sbaglio qualcosa o devo scaricare l'exe?

Raiken

SPE una volta installato produce dei file *.py*, è sufficiente lanciare *spe.py* per avviare il programma. Ovviamente bisogna avere installato Python. Se dovesse apparire per un momento il prompt di msdos e poi scomparire, molto probabilmente mancano alcune librerie essenziali. Prima di tutto *wxPython*, è sufficiente installarle scaricandole da *http://www.wxpython.org/*.

Certificare la sicurezza

In qualità di responsabile della rete di una grossa azienda mi rendo conto di assumermi rischi notevoli. Non che non abbia fiducia nella mia preparazione, tuttavia mi rendo conto che se qualcuno violasse la mia rete, sarebbe un grosso danno per la mia azienda. Avrei intenzione di stipulare un'assicurazione che mi copra da eventuali addebiti nel caso che capiti qualcosa di imprevisto. Per far questo, credo però che qualcuno dovrebbe certificare che il lavoro da me svolto sia corretto. Come posso fare?

Francesco

Il problema è sentito, allo stato attuale procurarsi un'assicurazione contro danni del genere potrebbe essere un'idea ma francamente non riesco a dire come la normativa assicurativa possa relazionarsi al problema. Posso invece suggerire di visitare *http://www.iscom.gov.it/ocsi.htm* ovvero il sito dell'OCSI - Organismo di certificazione della sicurezza di sistemi e prodotti nel settore della tecnologia della comunicazione e dell'informazione ICT – che si occupa appunto di certificare la sicurezza delle reti. Recentemente è anche stato pubblicato un volume che spiega a quali fattori attenersi per garantire che la propria configurazione sia adeguata alla rete da proteggere.

Che cos'è Plone?

Un mio amico mi ha detto che il miglior sistema di Content Management Systems al mondo è Plone. Ho cercato un po' in rete ma non ho trovato niente che mi spiegasse come funziona. Mi date qualche dritta?

Plone è un sistema di CMS basato su Zope che a sua volta è un application server basato su Python. I vantaggi sono innumerevoli, legati soprattutto alla gestione della sicurezza, ma anche alla facilità di inserimento delle informazioni, viceversa l'altro lato della medaglia è dovuto alla complessità del sistema che prevede un'installazione di Zope e un tempo di apprendimento di Zope stesso. In reatà questo non deve spaventare oltremodo, esistono infatti degli installer semplificati che mettono in grado anche i meno esperti di usare lo strumento. Per configurazioni professionali l'impegno richiesto è sufficientemente maggiore.

A chi spedire la posta?

Alla nostra redazione arriva spesso un considerevole numero di email riguardante temi specifici. Per consentirci di rispondere velocemente e in modo adeguato alle vostre domande abbiamo elaborato una FAQ – Frequently Ask Question – o risposte alle domande frequenti in italiano, di modo che possiate indirizzare le vostre richieste in modo mirato.

Problemi sugli abbonamenti

Se la tua domanda ha a che fare con una delle seguenti:

- Vorrei abbonarmi alla rivista, che devo fare?
- Sono un abbonato e non ho ricevuto la rivista, a chi devo rivolgermi?
- Sono abbonato ma la posta non mi consegna regolarmente la rivista, a chi devo rivolgermi?

Contatta **abbonamenti@edmaster.it** specificando che sei interessato a ioProgrammo. Lascia il tuo indirizzo email e indica il numero dal quale vorresti far partire l'abbonamento. Verrai contattato al più presto. Oppure puoi chiamare lo **02 831212**

Problemi sugli allegati

Se riscontri un problema del tipo:

- Ho acquistato ioProgrammo ed il Cdrom al suo interno non funziona. Chi me lo sostituisce?
- Ho acquistato ioProgrammo ma non ho trovato il cd/dvd all'interno, come posso ottenerlo?
- Vorrei avere alcuni arretrati di ioProgrammo come faccio?

Contatta **servizioclienti@edmaster.it**
Non dimenticare di specificare il numero di copertina di ioProgrammo e la versione: con libro o senza libro. Oppure telefona allo **02 831212**

Assistenza tecnica

Se il tuo problema è un problema di programmazione del tipo:

- Come faccio a mandare una mail da PHP?
- Come si instanzia una variabile in c++?
- Come faccio a creare una pagina ASP.NET

o un qualunque altro tipo di problema relativo a tecniche di programmazione, esplicita la tua domanda sul nostro forum: **http://forum.ioprogrammo.it**, uno dei nostri esperti ti risponderà. Le domande più interessanti saranno anche pubblicate in questa rubrica.

Problemi sul codice all'interno del CD

Se la tua domanda è la seguente

- Non ho trovato il codice relativo all'articolo all'interno del cd

Consulta la nostra sezione download all'indirizzo **http://cdrom.ioprogrammo.it**, nei rari casi in cui il codice collegato ad un articolo non sia presente nel cdrom, senza dubbio verrà reso disponibile sul nostro sito.

Idee e suggerimenti

Se sei un programmatore esperto e vuoi proporti come articolista per ioProgrammo, oppure se hai suggerimenti su come migliorare la rivista, se vuoi inviarci un trucco suggerendolo per la rubrica tips & tricks invia una email a **ioprogrammo@edmaster.it**

NEWSGROUP
Le informazioni nella Rete

Direttamente dal forum di ioProgrammo http://forum.ioprogrammo, le discussioni più "Hot" del momento

C#

Chiudere un'applicazione

Avrei la necessità di chiudere un'applicazione in relazione ad un evento, ad esempio se non è possibile leggere un file di configurazione. Come posso fare?

Forevry

http://forum.ioprogrammo.net/thread.php?threadid=5734&boardid=29

Risponde SalvatoreMeschini

Una soluzione percorribile è la seguente:

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace QuestoProgrammaVieneChiuso
{ public class MainForm :
                    System.Windows.Forms.Form
  {  public MainForm()
     { InitializeComponent(); }
     [STAThread]
     public static void Main(string[] args)
     { Application.Run(new MainForm());}
     private void InitializeComponent() {
        this.Load += new System.EventHandler(
                        this.MainFormLoad); }
     void MainFormLoad(object sender,
                        System.EventArgs e)
     { MessageBox.Show("Sto per chiudere
                              la finestra");
        Close(); } }
}
```

Java

Interrogare le porte USB

Come posso fare a utilizzare Java per pilotare periferiche connesse alle porte usb?

manu7377

http://forum.ioprogrammo.net/thread.php?threadid=5733&boardid=18

Risponde SalvatoreMeschini

Puoi controllare all'indirizzo *http://jusb.sourceforge.net/* dove viene rilasciata una Java Api OpenSource che fa esattamente al caso tuo.

C++

C++ problema con ShellExecute

Salve a tutti. Vorrei utilizzare il comando ShellExecute per lanciare il programma HTTrack da un mio personale programma scritto in c++ per scaricare automaticamente delle pagine Internet. Il problema è che devo inserire alcune opzioni oltre al nome del file da aprire come l'indirizzo della pagina e indicazioni per il download (dove salvare etc..) ma non funziona in nessun modo... In altre parole mi serve l'equivalente del comando DOS.

"C:\Programmi\web\WinHTTrack>httrack "http://spaceflightnow.com/cassini/050422titan.html" -O "c:\documenti\astronomia\news\tent_automatico" in c++. (Uso Dev c++).

Perché

http://forum.ioprogrammo.net/thread.php?threadid=5663&boardid=20

Rispondono blusterNrg e johnkoenig

Questo è un test eseguito con Wget

```
ShellExecute( NULL, "open", "c:\\wget.exe",
        "http://www.google.it -r", NULL, 1);
```

devi aggiungere il percorso e l'estensione al programma...

```
ShellExecute(NULL,"open","C:\\Programmi
        \\HTTTrack\\HTTrack.exe", "http:
        //spaceflightnow.com/cassini
        /050422titan.html -O c:\documenti
\astronomia\news\tent_automatico",NULL,1);
```

Riassumendo il comando giusto è:

```
ShellExecute(NULL,"open","C:\\Programmi
        \\web\\WinHTTrack\\HTTrack.exe",
        "http://spaceflightnow.com/cassini
        /050422titan.html -O /documenti
        /astronomia/news/tent_automatico
                                ",NULL,1);
```

Quante vocali ci sono in un file di testo?

Chi mi dà una mano ad implementare un algoritmo che conti le sole vocali all'interno di un file?

Leonardo12345

http://forum.ioprogrammo.it/thread.php?threadid=5696&boardid=20

Risponde domtes

Un metodo che utilizza del codice abbastanza leggibile è il seguente:

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
/**
 * Predicato che restituisce true quando
                      viene valutato su
 * una vocale
 */
bool isVowel(char c) {
  static string vowels("aeiou");
  return (string::npos !=
         vowels.find_first_of(tolower(c))); }
int main(int argc, char *argv[])
{ // Controllo parametri:
  if(2 != argc) {
    cerr << "Uso programma: cv file.txt\n";
    return 1; }
```

```
// Apertura file:
ifstream input(argv[1]);
if(!input) {
    cerr << "Impossibile aprire il file "
                              << argv[1] << '\n';
    return 1; }
string line;
int vowelCount = 0;
// Legge una riga alla volta ed effettua il
                                    conteggio:
while(getline(input, line)) {
    vowelCount += std::count_if(
        line.begin(), line.end(), isVowel); }
  cout << vowelCount << '\n';
    return 0;
}
```

VB.NET

Clipboard con testo da Word

Ciao. Io copio del testo da Word o da un altro editor di testo e quindi per recuperare questo evento da codice faccio come sotto:

```
Dim iData As IDataObject =
                    Clipboard.GetDataObject()
If iData.GetDataPresent(
                    DataFormats.Text) Then
zString = CType(iData.GetData(
                    DataFormats.Text), String)
end if
```

La mia domanda adesso è: se io volessi recuperare oltre il testo anche il font e il colore del testo che ho copiato da word esite un modo oppure no?
Grazie

AleTax

http://forum.ioprogrammo.it/thread.php?threadid=5578&boardid=27

Risponde sadeness

Devi usare *iData.GetData(DataFormats .Rtf)*, puoi mandarlo come output ad un richtextbox o metterlo in una variabile object. Un esempio che funziona è il seguente:

```
Dim prova As String = CStr(
            iData.GetData(DataFormats.Rtf))
RichTextBox1.Rtf = prova
RichTextBox1.SelectAll()
'Il colore
MsgBox(RichTextBox1.SelectionColor.ToString)
' Font Name
MsgBox(RichTextBox1.SelectionFont.Name)
' Font Size
MsgBox(RichTextBox1.SelectionFont.Size)
' Font Name - Size - ecc.
MsgBox(RichTextBox1.SelectionFont.ToString)
```

Gestire i ComboBox

Ciao a tutti, premetto che arrivo da VB6... Ho bisogno di aggiungere un elenco di persone in una combobox.
Al momento del click sull'item voglio ottenere un campo della classe persona, il codice ad esempio.(in VB6 usavo l'itemdata). Ho creato una classe Persona con campi Nome e Codice, per ogni persona che inserisco creo una nuova "Persona", setto i campi ed eseguo:

```
combobox.items.add(NuovaPersona)
```

a livello di programma funziona, inserisce l'intero oggetto nel combo da cui con il click posso recuperare i campi della classe... con un piccolo particolare: nel combo viene visualizzato un elenco di "System.qualcosa...", mentre sarebbe bello visualizzare uno dei campi, ad esempio il nome. Se inserisco solo il "da visualizzare" (Nome) anziché tutto l'oggetto non ho più gli altri campi, quindi non serve più a nulla nemmeno la classe... come posso fare?
Grazie a tutti.

marco881

http://forum.ioprogrammo.net/thread.php?threadid=5713&boardid=27

Risponde AmdBook

Devi impostare la proprietà *DisplayMember* del ComboBox su *"Nome".*
Un esempio di codice è il seguente:

```
Public Class persona
    Private _nome As String
   Private _cognome As String
Sub New(ByVal nome As String, ByVal
                              cognome As String)
    Me._nome = nome
    Me._cognome = cognome
End Sub
Public Property nome() As String
Get
    Return _nome
End Get
Set(ByVal Value As String)
_nome = Value
End Set
End Property
Public Property cognome() As String
Get
   Return _cognome
End Get
Set(ByVal Value As String)
    _cognome = Value
End Set
End Property
End Class
```

...codice associato al ComboBox:

```
ControlloComboBox.DisplayMember = "Nome"
ControlloComboBox.Items.Add(
                New persona("Pinco", "Pallino"))
```

Arrotondare migliaia

Vorrei arrotondare un numero come segue:
65893 ----> 60000.
Si può fare?
Nella classe math non ho trovato nessun metodo adatto!

whirp

http://forum.ioprogrammo.it/thread.php?threadid=5681&boardid=27

si risponde da solo

Ho risolto molto semplicemente, senza utilizzare metodi gia pronti:

```
If mutuo >= 100000 Then
mutuo = mutuo / 10000
mutuo = CInt(mutuo) * 10000
Else
mutuo = mutuo / 1000
mutuo = CInt(mutuo) * 1000
End If
```

arrotonda solamente valori superiori a mille, ad esempio

1244 = 1000
12433 = 12000
335400= 330000

Segreteria telefonica senza pagare un euro

Intercettiamo una telefonata, rispondiamo con un robot, chiudiamo il microfono e salviamo il messaggio del chiamante in formato audio sul telefono per riascoltarlo in un secondo momento!

Eccoci a parlare del sistema operativo più usato sui dispositivi portatili: il Symbian. In questo articolo il nostro obiettivo sarà la realizzazione di una segreteria telefonica che non abbia costi d'ascolto, da utilizzare come servizio aggiuntivo nel nostro cellulare. Il funzionamento di questa segreteria è piuttosto semplice. Viene intercettato lo squillo, viene spento il microfono del cellulare, il software risponde con un messaggio automatizzato, il chiamante lascia un messaggio che sarà registrato in formato audio sul cellulare. Per ascoltare il messaggio sarà sufficiente attivare il player del cellulare bypassando completamente il provider di servizi telefonici per quanto riguarda i costi d'ascolto.

SCHEMA PRINCIPALE DEL PROGRAMMA

Il nostro programma sarà composto da tre parti principali. La prima sarà quella che si occuperà di: intercettare la chiamata, aspettare un tot di squilli, chiudere il microfono onde evitare che, l'utente chiamante, capisca che siamo dall'altra parte del telefono. La seconda, provvederà a: rispondere alla chiamata, riprodurre un suono scelto dall'utente contenente un messaggio di risposta automatizzato. La terza ed ultima parte, fondamentale e fulcro di tutta l'applicazione, è la routine che si occuperà della registrazione dell'audio proveniente esclusivamente dalla linea. I passaggi elencati, non sono facili da realizzare e soprattutto non sono facilmente comprensibili. Perciò, per leggere quanto segue, dovrete dotarvi di pazienza e buona volontà, una volta superate le difficoltà iniziali, i risultati saranno entusiasmanti.

Fig. 1: **Alcuni fra i telefoni Symbian più venduti**

ALL'INTERNO DI SYMBIAN

Al solito il punto di ingresso dell'applicazione sarà *SimpleRecorderApp.cpp* contenuto nella directory *src*. Come in tutte le applicazioni Symbian che si rispetti, questo file non deve fare altro che creare il document dell'applicazione. Il document dell'applicazione non è che un ponte verso l'interfaccia utente. In questa sezione vengono tipicamente inserite tutte le funzioni che servono alla gestione dei file. Il tutto, nel nostro caso viene definito in *SimpleRecorderDocument.cpp,* è qui che viene richiamata la gestione dell'interfaccia utente, contenuta nel file *SimpleAppRecorderUI.cpp,* è questo file che include *RecordEngine.h. RecordEngine.cpp* conterrà una intera classe atta alla gestione della segreteria telefonica così come l'abbiamo descritta ad inizio articolo.

COME INIZIARE

È necessario avere installato il Nokia Sdk relativo al proprio cellulare. Previa registrazione potete ottenerlo all'indirizzo http://www.forum.nokia.com/main/0,6555,034-4,00.html. È necessario anche avere installato una copia di ActivePerl, potete scaricarlo all'indirizzo http://www.activestate.com. Infine è necessario avere installato Microsoft Visual Studio 2003. L'articolo chiaramente presuppone qualche base per quanto riguarda la programmazione dei cellulari basati sul sistema operativo Symbian e del Nokia SDK.

REQUISITI

Conoscenze richieste
Conoscenza approfondita di Symbian e C++

Software
.NET Framework, Visual Studio 2003

Impegno

Tempo di realizzazione

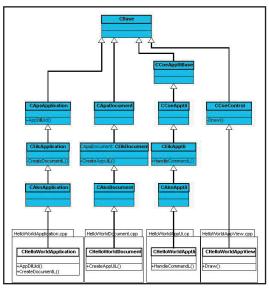



Fig. 2: ***Schema dell'applicazione Symbian HelloWorld***

Per una comprensione più completa di com'è strutturata un'applicazione Symbian, vi rimandiamo alla **Figura 2**, dove c'è lo schema basato sull'esempio fornito nell'SDK, Hello World.
Diamo uno sguardo a *RecordEngine.h*:

```
class CRecorderEngine: public
                    MMdaObjectStateChangeObserver {
  public:
  virtual void MoscoStateChangeEvent(CBase*
    aObject, TInt aPreviousState, TInt aCurrentState,
                                   TInt aErrorCode);
  public:
    static CRecorderEngine * NewL(const TDesC&);
    static CRecorderEngine * NewLC(const TDesC&);
    long   StartRecordL();
    void   StopRecord();
[...]
  };
}
```

L'engine è basato su due funzioni principali che sono la *StartRecordL()* e la *StopRecordL()*.
In questa prima fase ci interessa capire il funzionamento della *StartRecordL()* in cui inseriremo le istruzioni per gestire la telefonata.

DUE PAROLE SULL'EREDITARIETÀ

Quanto diremo qui di seguito potrebbe apparire piuttosto astratto e concettualmente difficile da recepire. Potete saltare questo paragrafo e leggerlo successivamente se siete interessati alla pura pratica, altrimenti con un po' di sforzo potete acquisire qualche informazione in più su alcune scelte architetturali della programmazione Symbian. Per utilizzare il sottosistema audio, dobbiamo rendere la nostra classe, derivante dall'interfaccia *MMdaObjectStateChangeObserver.* Questo è l'unico modo per poter utilizzare il sottosistema audio, dato che il Symbian gestisce in modo particolare l'ereditarietà delle classi. Com'è possibile vedere dalla documentazione ufficiale, esse derivano da un'unica classe e cioè la *CBase*. La dicitura delle classi è praticamente simile a quella utilizzata dal Java:

- **Classe C** - Classe derivata dalla *CBase* ed istanziata in memoria.
- **Classe T** - Classi con assenza di oggetti esterni.
- **Classe M** - Classi di interfaccia.
- **Classe R** - Classi con collegamenti a risorse reali.

Ricordiamo che nel sistema dell'ereditarietà delle classi in Symbian, la derivazione di una classe *C*, dev'essere dichiarata per prima e che, da una classe *C*, può derivare una sola classe *C* e nessuna o più classi *M*. Inoltre non possiamo derivare contemporaneamente da due classi *M*, una classe *C*, se la prima classe *M* deriva direttamente dalla seconda.
Per capire meglio il concetto, riferitevi alla **Figura 3.**
Tornando all'interfaccia *MMdaObjectStateChangeObserver*, dobbiamo sapere che c'è un solo metodo al suo interno, e precisamente il *MoscoStateChangeEvent()*, al quale vengono passati da Symbian, una serie di parametri e di valori che non dobbiamo usare direttamente.

- **CBase* aObject** - Il puntatore alla struttura *CBase* (all'*Audio Sample Object*) che gestisce i

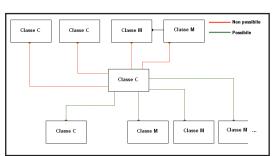



Fig. 3: ***Regole di ereditarietà in Symbian***

MA FUNZIONA A TELEFONO SPENTO?

Ovviamente no. Questo non perché alcune applicazioni symbian non possano funzionare a telefono spento. Vedi ad esempio la sveglia o il calendario, ma perché molto semplicemente l'assenza della portante ci impedisce di intercettare la telefonata.
Alcuni lettori potrebbero far notare che i cellulari di ultima generazione, hanno la possibilità di impostare la sveglia anche a terminale spento, questo è possibile perché l'alimentazione minima, riesce ad fornire, in parole povere, energia sufficiente per far sì che un evento, come quello della sveglia, accenda il telefono. Per la nostra segreteria questo risulterebbe impossibile dato che l'unico evento che interessa noi, è l'avvento di una chiamata in arrivo e, come detto prima, non essendo connessi alla rete, non si verificherebbe mai.

NOTA

Il programma è stato testato e compilato con il Microsoft Visual C++ e con l'SDK della Nokia compatibile con il Nokia 6670.
La scelta per il Microsoft Visual C++ è dovuta alla sua diffusione ed anche per permettere al lettore di integrarsi alla perfezione con il corso iniziato sulle pagine di IoProgrammo.

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

cambiamenti e i vari stati.

- **TInt aPreviousState** - Contiene lo stato precedente.
- **TInt aCurrentState** - Contiene il codice di stato attuale.
- **TInt aErrorCode** - Contiene il codice di errore se, il suo valore, è *false*.

Quindi, riepilogando, per gestire un sottosistema audio, bisogna passare un puntatore della classe *CBase* al metodo *MoscoStateChangeEvent()*. Per far questo, la classe principale che sfrutta l'adattatore audio, deve derivare dall'interfaccia *MMdaObjectStateChangeObserver* che a sua volta, deriva dalla classe principale *CBase* come vogliono le specifiche Symbian. Questo vale per tutte le classi di interfaccia contrassegnate dalla dicitura *M*.

INIZIAMO A REGISTRARE

Come già detto, all'interno della classe *CRecorderEngine* la funzione per noi più importante è la *StartRecordL* che si occupa di dare il via alla registrazione della telefonata in arrivo. Questa funzione al suo interno, utilizza varie API che fanno al caso nostro e che il Symbian ci mette a disposizione, dando uno sguardo a RecordEngine.cpp tutto sarà più chiaro. All'interno di *RecordEngine .cpp* troviamo

```
CRecorderEngine ::StartRecordL(){
...
m_line.NotifyIncomingCall(m_iStatus,m_newCallName);
User::WaitForRequest(m_iStatus);//  waiting for
                                       incoming call
User::LeaveIfError(m_call.OpenExistingCall(m_line,
                                         m_newCallName));
User::LeaveIfError(m_call.AnswerIncomingCall());
```

Come possiamo notare nel codice riportato, compaiono delle API che si occupano dell'attesa e della risposta di una chiamata in arrivo. Utilizzando queste API si riesce a gestire nei minimi particolari l'intero telefono. Le API che andremo ad utilizzare, sono usate dalla stragrande maggioranza dei telefoni che utilizzano il Symbian come sistema operativo.

Fig. 4: *I colossi che si avvalgono del Symbian per il loro cellulari*

È anche importante sottolineare che quando riceviamo una chiamata, il nostro device viene considerato *Client*, mentre il *Device* che effettua la chiamata, viene considerato *Server*. Questo fa capire che la gestione della chiamata avviene proprio come se si stesse effettuando una connessione di rete tra un *Server* ed un *Client*. La prima funzione notevole all'interno del codice esposto è la *NotifyIncomingCall()* che si occupa di notificare al Client, la chiamata in arrivo. Questa funzione è definita nella classe *RLine* che troviamo nel file di inclusione *etel.h*. Seguendo linea per linea, troviamo un'altra funzione, e precisamente la *WaitForRequest()*. La funzione citata, non fa altro che attendere la chiamata entrante in modo da generare un evento *(asyncronous request)* appena la chiamata è stata ricevuta. La funzione *WaitForRequest()* la si può trovare nella definizione della classe *User.* Andando sempre avanti nella lettura del codice, troviamo la funzione *OpenExistingCall()* che apre il registro chiamate del Server, stabilendo così il canale di comunicazione.

STRUTTURA DI UN'APPLICAZIONE SYMBIAN UTILIZZANDO IL WIZARD

- **aif** - Sono contenute le immagini bmp, le icone e il file RSS delle risorse.
- **data** - Altri file RSS che comprendono la gestione dei dati in ingresso e la posizione e l'utilizzo delle voci che compaiono nei menu standard del telefono.
- **Engine** - Contiene il codice e la classe principale.
- **group** - Contiene tutti i files per la compilazione e la generazione del package
- **inc** - Contiene gli #include principali e il file .loc, utilizzato per la localizzazione del programma e il file .hrh che contiene la definizione delle costanti.
- **install** - Contiene il Package (.pkg) e il .SIS una volta ottenuto con il batch
- **src** - Contiene i sorgenti principali di un'applicazione Symbian.

Per poter ottenere il tutto, bisognerà entrare con la shell DOS nella cartella group del nostro progetto ed eseguire il Batch: "bldmake bldfiles" e successivamente "ebld build thumb urel". Una volta fatto questo, passeremo alla creazione del file .SIS mediante l'esecuzione della seguente linea: "makesis simplerecord.sis"

Interconnessione tra i files

Scorrendo ancora il codice, arriviamo all'ultima funzione riportata nelle righe precedenti, e cioè la funzione *AnswerIncomingCall()*. Questa funzione si occupa di, una volta "svegliata" dal segnale di ricevimento della chiamata, rispondere eseguendo un'alzata di "cornetta".

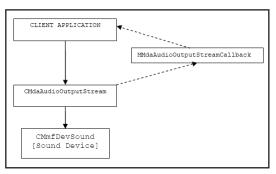



Fig. 5: ***Interazione delle classi per lo streaming audio***

La *AnswerIncomingCall()* è definita nella classe *RCall*, sempre nel file di inclusione *etel.h* Continuando ad analizzare il codice contenuto nella funzione membro *StartRecordL()*, ci troviamo di fronte alla serie di istruzioni che eseguono la registrazione audio:

```
…
m_iMdaAudioRecorder->SetAudioDeviceMode(
    CMdaAudioRecorderUtility::ETelephonyNonMixed);
m_iMdaAudioRecorder->SetGain(
                m_iMdaAudioRecorder->MaxGain());
m_iMdaAudioRecorder->SetPosition(
                TTimeIntervalMicroSeconds(0));
m_iMdaAudioRecorder->CropL();
m_iMdaAudioRecorder->RecordL();
m_iMdaAudioRecorder->SetVolume(
                m_iMdaAudioRecorder->MaxVolume());
return 0;
}
```

Riassumendo la *StartRecordL()* esegue i seguenti compiti che sono evidenti dal codice

- Notifica al sistema operativo che sta per arrivare una chiamata.
- Si mette in attesa della chiamata in arrivo.
- Se tutto va a buon fine apre un canale di comunicazione.
- Se tutto va a buon fine rispondere.
- Registra quanto viene detto all'altro capo del telefono da chi sta chiamando.

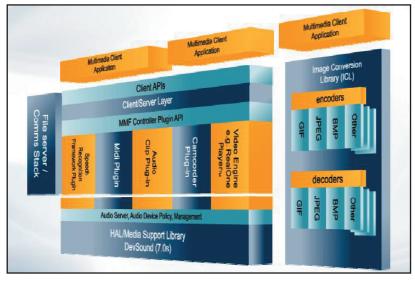
REGISTRAZIONE DEI SUONI

Per poter gestire la registrazione di un suono, dobbiamo avvalerci della classe *CmdaAudioRecorderUtility.* Questa classe è fondamentale sia per la riproduzione di suoni sia della loro registrazione. *CmdAudioRecordereUtility* è definita nel file di inclusione *MdaAudioSampleEditor.h* e una volta allocata, ci metterà a disposizione una serie di funzioni membro. Come possiamo notare, per impostare una corretta registrazione che proviene esclusivamente dalla linea, abbiamo bisogno della funzione *SetAudioDeviceMode()*. Questa funzione, necessita come parametro, il tipo *TDeviceMode* che permette di impostare appunto il device audio nei seguenti modi:

- *EDefault*
- *ETelephonyOrLocal*
- *ETelephonyMixed*
- *ETelephonyNonMixed*
- *ELocal*

Degli *Enum* elencati, abbiamo utilizzato il solo *ETelephonyNonMixed* che rispetto all'*ETelephonyMixed*, registra l'audio proveniente esclusivamente dalla linea, senza effettuare anche la registrazione di audio proveniente dal microfono del nostro telefono. Scendendo ancora nel codice, troviamo la funzione *SetGain()*, che imposta il guadagno dell'audio, e la funzione *SetPosition()*, che viene utilizzata per far partire la registrazione dall'inizio. La funzione *CropL()* è utilizzata per far sì che la zona adibita all'audio, venga per prima azzerata e successivamente con l'utilizzo della funzione *RecordL()*, verrà riempita con l'audio proveniente dalla linea. Infatti il metodo *RecordL()*, si occupa di concatenare allo stream già presente, quello che la linea sta inviando al Client, ecco il perché dell'utilizzo della funzione *CropL()* prima di eseguire la registrazione. Una volta iniziata la registrazione, possiamo impostare il volume al massimo chiamando la funzione: *SetVolume()* che, come parametro, avrà direttamente la funzione *MaxVolume()*. Questo perché la funzione *MaxVolume()*, restituisce il valore che corrisponde al massi-

Fig. 6: ***Schema delle API***

Ci sono diversi RAD che assicurano un interfacciamento perfetto con i devices Symbian. Sicuramente il nostro consiglio cade sui prodotti Borland, azienda che con il suo C++ BuilderX Mobile Version, ha lasciato una firma indelebile anche nel campo della programmazione per la telefonia mobile su Symbian.

mo del volume permesso dal device e quindi avendola utilizzata come parametro della funzione *SetVolume()*, imposterà il massimo valore consentito. Una funzione membro della classe *CRecorderEngine,* di importanza vitale, è quella adibita alla costruzione delle classi sulla gestione del file audio, e delle impostazioni audio che effettuano la registrazione. Tutto questo è contenuto nella funzione *CRecordEngine::CostructL()*, di cui vediamo il codice:

```
void CRecorderEngine::ConstructL(const TDesC&
                                        wav_file_name) {
  m_file_name.Copy(wav_file_name);
  m_iMdaAudioRecorder =
      CMdaAudioRecorderUtility::NewL(*this,NULL,80);
  TMdaAudioDataSettings iSettings;
  iSettings.iChannels = 1;
  iSettings.iSampleRate = 8000;
  m_iMdaAudioRecorder->OpenL(new TMdaFileClipLocation(
          m_file_name),new TMdaWavClipFormat(),new
                        TMdaPcmWavCodec(),&iSettings);
}
```

Leggendo il codice, notiamo l'utilizzo della funzione *.Copy,* metodo ottenuto mediante la dichiarazione di tipo *TBuf* che troviamo nel file di inclusione *RecorderEngine.h.* Quello che bisogna però osservare con maggiore attenzione, sono i settaggi per la classe *TMdaAudioDataSettings,* che servono ad impostare i parametri base per l'utilizzo e l'apertura del media server. È possibile trovare la definizione di questa classe nel file di inclusione *audio.h.* Questi parametri permettono di impostare la risoluzione di campionamento, il numero di canali da applicare al sample audio, il volume, e altre impostazione che, per maggior completezza, consigliamo di visionare direttamente sulla guida ufficiale. Come ultima funzione, troviamo la funzione *OpenL()* che viene utilizzata quando si vuole aprire un oggetto audio da riprodurre o quando si vuole creare un oggetto audio per la sua registrazione. Il nome effettivo del file di registrazione, è definito nel file *SimpleRecorderAppUI.cpp,* con la seguente linea:

```
_LIT(KDefaultFileName, "c:\\voice.wav");
```

Ciò che abbiamo analizzato in questa sezione, è il principale corpo, quello che più ci interessa, della segreteria telefonica stand-alone.

Per poter creare un'applicazione Symbian con il Visual Studio C++ 6, bisogna avvalersi del Wizard che troviamo nell'SDK fornito dalla Nokia e appositamente studiato per il vostro cellulare. Copiare lo stesso, nella directory Template del Visual Studio e poi, avvalersi del pannello Wizard del Visual Studio che è possibile raggiungere dal menu "New".

RISPONDERE CON UN FILE AUDIO E DISATTIVARE IL MICROFONO

Accingiamoci dunque ad aggiungere quello che consideriamo una rifinitura all'esempio allegato. La prima cosa da effettuare è quella senza alcun dubbio, di disattivare il microfono. Purtroppo, solo dalle versioni superiori del Symbian, è possibile effettuare registrazioni tramite linea e disattivare il microfono tramite apposite API messe a disposizione dello sviluppatore. Per i telefoni precedenti ma sempre corredati di sistema operativo Symbian, è necessario utilizzare un trucco; il trucco è quello di "simulare" la pressione del tasto *Mute,* che compare quando c'è una chiamata in corso. Il codice è relativamente semplice, diamogli uno sguardo:

```
void CSimpleRecordAppUIi::MuteCallL(){
  RWsSession sess=CCoeEnv::Static()->WsSession();
  TWsEvent event;
  TInt id=sess.FindWindowGroupIdentifier( 0,
                                     _L("*Phone?") );
  event.SetType(EEventKey);
  event.SetTimeNow();
  event.Key()->iCode = EKeyCBA1;
  event.Key()->iModifiers = 0;
  event.Key()->iRepeats = 0;
  event.Key()->iScanCode = EStdKeyNull;
  sess.SendEventToWindowGroup( id, event );
  event.SetType(EEventKey);
  event.SetTimeNow();
  event.Key()->iCode = EKeyDownArrow;
  event.Key()->iModifiers = 0;
  event.Key()->iRepeats = 0;
  event.Key()->iScanCode = EStdKeyNull;
  sess.SendEventToWindowGroup( id, event );
  event.SetType(EEventKey);
  event.SetTimeNow();
  event.Key()->iCode = EKeyDownArrow;
  event.Key()->iModifiers = 0;
  event.Key()->iRepeats = 0;
  event.Key()->iScanCode = EStdKeyNull;
  sess.SendEventToWindowGroup( id, event );
  event.SetType(EEventKey);
  event.SetTimeNow();
  event.Key()->iCode = EKeyOK;
  event.Key()->iModifiers = 0;
  event.Key()->iRepeats = 0;
  event.Key()->iScanCode = EStdKeyNull;
  sess.SendEventToWindowGroup( id, event );
}
```

Come avete potuto notare, il codice è semplicissimo da intuire, poiché non fa altro che generare una serie di eventi sui tasti che verranno poi inviati al device. Nel nostro caso, verrà eseguito quando la telefonata è stata intercettata ed è stata chiamata la funzione *AnswerIncomingCall()*. Per ripristinare poi lo stato di default e quindi il microfono attivo, ma non è il nostro caso dato che la comunicazione verrà interrotta una volta registrato il suono, bisognerà rieseguire la serie di eventi, modificandoli in modo da adattarli alla sequenza di voci del menu del proprio

device. Il codice potrà essere aggiunto al sorgente della nostra applicazione o aggiunto alla classe principale facendolo diventare un metodo e quindi, ritoccando la *CSimpleRecordAppUi*. Questa classe si occupa della gestione degli eventi della tastiera. Facendo così, non dovremo far altro che richiamare poi la funzione all'occorrenza. È possibile inserirla, ed è consigliabile, anche direttamente nella funzione *MoscoStateChangeEvent* in questo modo:

```
void CRecorderEngine ::MoscoStateChangeEvent(
        CBase* aObject, TInt aPreviousState, TInt
                aCurrentState, TInt aErrorCode){
 if(aCurrentState == CMdaAudioClipUtility::ERecording){
 // sorgente per il mute
 }else if (aErrorCode != KErrNone) {
 StopRecord();}
}
...
```

Dopo il microfono, la nostra segreteria avrà bisogno di una routine che si occuperà di aspettare un tot di squilli, prima di stabilire la connessione.
Il codice di attesa che occorrerà, è il seguente:

```
void current_thread :: sleep(DWORD timeout){
 TInt iMSec;
 iMSec = timeout*1000;
 RTimer timer;
 TRequestStatus timerStatus;
 timer.CreateLocal();
 TTimeIntervalMicroSeconds timeIntervalMS(iMSec);
 timer.After(timerStatus,iMSec);
 User::WaitForRequest(timerStatus);
 }
```

Il Symbian purtroppo non ci viene in aiuto per quanto concerne il calcolo degli squilli ricevuti, ma ci consente di stabilire il tempo di attesa prima della risposta. Questo piccolo esempio, fa capire come bisogna attendere un certo tempo, quando è in atto una chiamata in arrivo *(Incoming Call)*. Purtroppo non esistono alri mezzi se non quello di calcolare un minimo tempo prima di iniziare la registrazione o eventualmente riprodurre un suono di benvenuto. Anche per questo sorgente vale lo stesso discorso fatto per il precedente. Lasciamo al lettore la scelta di inglobarlo nella classe principale utilizzandolo come metodo, o semplicemente copiandolo sempre nella funzione *MoscoStateChangeEvent*.

RIPRODURRE UN SUONO PER LA RISPOSTA

Come ultima aggiunta, dobbiamo consentire alla segreteria telefonica la possibilità di rispondere alla chiamata riproducendo un suono pre-registrato di benvenuto. Affinché ciò sia possibile, bisogna creare un oggetto *CMdaAudioRecorderUtility*, successivamente avvalerci della funzione *OpenL()*, in modo da poter aprire il file che vogliamo utilizzare come benvenuto. La sintassi sarà così:

```
OpenL(new TMdaFileClipLocation(m_file_name), new
    TMdaWavClipFormat(), new TMdaPcmWavCodec(),
                                   &iSettings);
```

Successivamente, una volta aperta la connessione con la funzione *AnswerIncomingCall()* ed aver disattivato il microfono con la funzione precedentemente illustrata, dovremo chiamare la funzione *PlayL()*, metodo dell'oggetto appena destritto.

I VARI SDK

È necessario scaricare l'SDK associato e sviluppato esclusivamente per il telefono in vostro possesso. Alcune case consigliano agli sviluppatori, di avvalersi dell'SDK distribuito dalla Nokia mentre altre, distribuiscono sui loro siti web, SDK proprietari e messi a disposizione gratuitamente agli sviluppatori. È consigliabile quindi scaricare sempre SDK specifici per il telefono che si possiede e su cui si intende sviluppare l'applicazione.

CONCLUSIONI

Non mi stancherò mai di dire quanto sia flessibile questo sistema operativo per la telefonia mobile, e quanto sia degnamente supportato da colossi come Nokia, Sony-Ericsson. L'espandibilità, la programmabilità e la versatilità ne fanno, a parer mio, il miglior sistema operativo per telefonia mobile mai creato poiché permette agli sviluppatori, di aggiungere nuove funzionalità non presenti di default, e agli utenti, di sfruttarle e di averne sempre nuove. Con questo articolo vi abbiamo fornito le principali conoscenze per la gestione di una segreteria telefonica basata sul proprio cellulare invece che su un servizio offerto dal provider. Ovviamente l'applicazione può essere enormemente sviluppata, ad esempio inserendo delle funzioni per selezionare il messaggio di risposta, oppure per effettuare una *select* sulla base di un numero telefonico. Tutto deriva da quanto letto in questo articolo. Inviate pure le creazioni più fantasiose all'indirizzo *ioprogrammo@edmaster.it* A chi è neofita di questo tipo di programmazione, consigliamo di seguire il corso di "ioProgrammo" sulla programmazione Symbian, che da nozioni fondamentali quali la compilazione delle applicazioni, ed entrare molto più facilmente in quello che è, a mio giudizio, un mondo sempre più florido.

Marcello Scala

SUL WEB

http://www.forum.nokia.com
Dov'è possibile reperire tutti gli SDK.

http://www.symbian.com
Sito ufficiale del sistema operativo Symbian.

WEB Service Usiamoli da PHP

Muoviamo i primi passi nei web service. Vediamo come utilizzare un servizio esistente per realizzare qualcosa di divertente, ed infine creiamone uno tutto per noi

Molti di voi avranno sentito parlare di Web Service; molti sapranno anche perfettamente cosa sono e come funzionano, sicuramente in non molti avranno provato ad usarli o addirittura implementarne uno. In questo articolo assolveremo a diverse funzioni. Prima di tutto, diremo qualcosa su cosa sono i Web Service. In secondo luogo impareremo come usarli. infine creeremo un nostro servizio pronto per essere utilizzato e, perché no?...venduto.

COSA SONO I WEBSERVICE

L'idea di base è realizzare "applicazioni distribuite". Supponete di stare sviluppando un'applicazione di "prenotazione viaggi" per un'agenzia che ve ne ha fatto richiesta. Create un fantastico software che interroga tutti i tour operator del mondo, trova un albergo soddisfacente, effettua la prenotazione e infine stampa fattura e indicazioni per il viaggio con tanti saluti ai clienti. Contemporaneamente dall'altra parte del mondo un programmatore abile quanto voi riceve una commessa per lo stesso tipo di software. Anche lui inizia da capo, scrive milioni di righe di codice diverse dalle vostre per raggiungere il vostro stesso identico risultato. È abbastanza frustrante che si debba sempre reinventare l'acqua calda! Ora supponiamo che la vostra applicazione di prenotazione alberghiera si chiami *"TheBestBooking"*. Sarebbe bellissimo se il tizio dell'altra parte potesse scrivere nel proprio programma:

```
risultato = TheBestBooking->prenotaAlbergo(
                    "Roma","5","10/07/05","15/07/05");
```

Che cosa vuol dire questa riga? Vuol dire che invece di riscrivere da zero tutto il codice per realizzare la propria applicazione, molto semplicemente il software interroga un servizio messo a disposizione in rete da qualcuno che lo ha già realizzato. Questo consentirebbe a chi scrive il programma di evitare di dover perdere tempo con la creazione di funzionalità base, ed invece potersi concentrare sulla personalizzazione richiesta dai clienti, realizzando applicazioni più funzionali in meno tempo. *TheBestBooking* è un Web Service, ovvero un'applicazione che espone funzionalità sul Web.
Perché rendere disponibile il proprio lavoro al mondo? qualcuno potrebbe farlo semplicemente per "spirito di condivisione", per quella volontà di condividere le proprie scoperte al fine di migliorare la qualità della vita, altri potrebbero "vendere" i propri Web Service.

PHP5 VS PHP4

L'intero articolo si basa sull'utilizzo di PHP5. In PHP 4, l'uso dei Web Service si basava su librerie esterne quali ad esempio Nusoap, che utilizzano una sintassi e un approccio profondamente diversi da quelli qui discussi. Quindi, è fondamentale fare i vostri esperimenti utilizzando PHP5. Il secondo requisito essenziale è l'attivazione dell'estensione SOAP. Potete attivare l'estensione in ambito Linux compilando con il parametro *–enable-soap*, oppure con windows decommentando la linea *extension=php_soap .dll*, nella sezione relativa alle estensioni.

UTILIZZIAMO UN WEBSERVICE DA PHP 5

Prima di tutto stabiliamo quale Web Service utilizzare. In rete esistono tanti motori di ricerca per Web Service. Per il nostro primo esempio abbiamo utilizzato *http://uddi.microsoft.com*. Fra i tanti Web Service esposti ne abbiamo trovato uno che ci

restituisce informazioni sullo storico dei risultati dei vari tornei del Palio di Siena. Il WebService in questione è descritto all'indirizzo: *http://www.ilpalio.siena.it/Palio.asmx*. Delle varie righe di descrizione quello che ci interessa maggiormente in questo momento è il link al file WSDL che descrive il servizio. Nel nostro caso è *http://www.ilpalio.siena.it/Palio.asmx? WSDL*.
Cliccando sul link vi accorgerete che si tratta di un file XML. I file WSDL descrivono nel dettaglio i metodi e le proprietà esposte da un Web Service. Poiché WSDL è un formato standard appare ovvio che descrivere in questo modo un web service, consente a chiunque di comprendere quali sono i metodi che esso espone e come usarli. Diamo uno sguardo al contenuto del nostro file WSDL:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    " xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/e
    ncoding/" xmlns:tns="http://www.ilpalio.siena.it/"
        xmlns:tm="http://microsoft.com/wsdl/mime
                                   /textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
     targetNamespace="http://www.ilpalio.siena.it/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="http://www.ilpalio.siena.it/">
      <s:element name="Palii">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1"
                        name="Anno" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="PaliiResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
                        name="PaliiResult" type=
                        "tns:ArrayOfDettaglioPalio" />
[...]
```

Vi abbiamo riportato un estratto, ma diciamo che al momento non è facilissimo capire in questo modo cosa dovremmo fare per usare il nostro web service, anche se forse i più esperti avranno già intuito come potrebbe funzionare.
Ma proprio perchè WSDL è un formato standard, PHP è dotato di funzioni in grado di parserizzare il file WSDL in questione e restituirci informazioni corrette.
La nostra pagina PHP sarà così composta:

```
<?
  $client = new SoapClient(
      'http://www.ilpalio.siena.it/Palio.asmx?WSDL');
  echo "<pre>";
  $info=$client->__GetFunctions();
  print_r($info);
  echo "<hr>";
  $info=$client->__GetTypes();
  print_r($info);
  echo "</pre>";
?>
```

Con la prima istruzione creiamo un nuovo oggetto di classe *SoapClient*. Nel costruttore passiamo come parametro il file WSDL che contiene la descrizione del Web Service da utilizzare. A questo punto il file viene parserizzato, e l'oggetto *client* viene "riempito" con i metodi e le proprietà esposte dal Web Service. Inoltre all'oggetto client apparterranno alcuni metodi che ci consentono di "gestire" il Web Service in questione. Ad esempio il metodo *__GetFunctions()* restituisce l'elenco di tutti i metodi esposti. Nel nostro caso:

```
Array
(
[...]
  [8] => ArrayOfDettaglioPalio Palii(string $Anno)
  [9] => DettaglioPalio dettaglioPalio(string $Giorno)
  [10] => DettaglioPalio UltimoPalioVinto(string
                                        $Contrada)
  [11] => ArrayOfDettaglioPalio PaliiVinti(string
                                        $Contrada)
)
```

Abbiamo volutamente omesso alcune informazio-

Utilizza questo spazio per le tue annotazioni

WEB SERVICE SENZA WSDL

In questo articolo abbiamo sempre fatto riferimento alla lettura del file WSDL per ottenere i metodi esposti dal Web Service. Esiste un'altra tecnica che consente di non dover leggere il file WSDL ogni volta che si richiama il servizio. La tecnica in questione fa riferimento all'uso delle funzioni __call. Ha il vantaggio di non dovere parserizzare il file WSDL per ogni chiamata, ma lo svantaggio di essere meno semplice da utilizzare. D'altra parte all'interno del file php.ini trovate le seguenti linee:

```
[soap]
; Enables or disables WSDL caching feature.
soap.wsdl_cache_enabled=1
; Sets the directory name where SOAP extension will put cache files.
soap.wsdl_cache_dir="/tmp"
; (time to live) Sets the number of second while cached file will be used
; instead of original one.
soap.wsdl_cache_ttl=86400
```

Che consentono di abilitare o meno una cache per i file WSDL diminuendo ovviamente il tempo d'accesso.

ni che in questo momento non ci interessavano. Ma a questo punto l'elenco dei metodi esposti dal Web Service è sufficientemente chiaro. Ad esempio il metodo *Palii* prende in input una stringa e restituisce una struttura di classe *ArrayDettaglioPalio*. Ci rimane da stabilire come questa struttura viene definita. Per ottenere questo tipo di informazione ricorriamo al metodo *__GetTypes();* che infatti ci restituisce:

```
Array
(
 [2] => struct ArrayOfDettaglioPalio {
            DettaglioPalio DettaglioPalio;
 }

 [3] => struct DettaglioPalio {
 string Contrada;
 string Cavallo;
 string Fantino;
 [...]
 boolean Straordinario;
 boolean OrdineAlCanapeIncerto;
 boolean Scosso;
}
)
```

Anche in questo caso per brevità abbiamo omesso qualche informazione, ma il senso è chiaro. Non ci rimane che andare ad usare il nostro Web Service. L'esempio è il seguente:

```
<?
   $client = new SoapClient(
         'http://www.ilpalio.siena.it/Palio.asmx?WSDL');
   $Anno="1984";
   $SoapParam = array('Anno' =>$Anno);
   $Palii=$client->Palii($SoapParam);
   print ("<pre>");
   print_r($Palii);
   print ("</pre>");
?>
```

Che ci restituisce che nel 1984 nelle due tornate del 02/07e del 16/08 hanno vinto rispettivamente la contrada dell'Oca con Baiardo IV e la contrada del Nicchio con Orion. Nell'output completo c'è anche una descrizione della gara.
Volendo affinare un po' il nostro codice, potremmo trasformarlo in qualcosa del genere:

```
<?

   $client = new SoapClient(
         'http://www.ilpalio.siena.it/Palio.asmx?WSDL');
            $Anno="1984";
   $SoapParam = array('Anno' =>$Anno);
   $Palii=$client->Palii($SoapParam);
   foreach($Palii->PaliiResult->DettaglioPalio as
                                              $risultato) {
   print "Il ".$risultato->Data." ha vinto la contrada ".
   $risultato->Contrada." con ".$risultato->
                                        Fantino."<br>";
   };
?>
```

Osservate che abbiamo passato il parametro definendo un array il cui indice è identico al parametro richiesto dal metodo.

CREARE UN WEB SERVICE

Ovviamente possiamo anche esporre un Web Service, perché altri ne fruiscano. Considerate la seguente applicazione PHP:

```
<?
$copertina=array(
            "91"=>"La grafica costruita intorno a te",
            "90"=>"Parola di Java",
            "89"=>"Chat Bluetooth"
);
function GetCopertina($issue) {
global $copertina;
return $copertina[$issue];
}
ini_set("soap.wsdl_cache_enabled", "0");
$server = new SoapServer("dummysoap.wsdl");
$server->addFunction("GetCopertina");
$server->handle();
?>
```

Certamente non si tratta di un grande programma PHP. Ma quel che importa è capire che abbiamo appena definito il nostro primo Web Service. Come si può notare, la sintassi è banale. La cosa più importante è definire il file *dummysoap.wsdl* che contiene la descrizione del servizio. Riportiamo di seguito uno spezzone di codice che potete usare come template:

```
<?xml version ='1.0' encoding ='UTF-8' ?>
<definitions name='StockQuote'
targetNamespace='http://localhost/ioProgrammo/soap'
  xmlns:tns=' http://example.org/StockQuote '
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
           xmlns:soapenc='http://schemas.xmlsoap.org
                                       /soap/encoding/'
  xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
  xmlns='http://schemas.xmlsoap.org/wsdl/'>
<message name='getCopertinaRequest'>
  <part name='issue' type='xsd:string'/>
</message>
```

WEB SERVICE NOTEVOLI

Microsoft commercializza il proprio MapPoint esponendo le funzionalità dell'applicazione come Web Service. Tradotto in parole povere la vostra applicazione può interrogare MapPoint Web Service per conoscere ad esempio il percorso stradale da compiere per arrivare da un luogo ad un altro. Ciascuna interrogazione ha ovviamente un costo per chi la effettua; date uno sguardo http://www.microsoft.com/mappoint/products/webservice/default.mspx **per saperne di più.**

```
<message name='getCopertinaResponse'>
  <part name='Result' type='xsd:string'/>
</message>
<portType name='GetCopertinaPortType'>
  <operation name='getCopertina'>
    <input message='tns:getCopertinaRequest'/>
    <output message='tns:getCopertinaResponse'/>
  </operation>
</portType>
<binding name='GetCopertinaBinding' type=
                                  'tns:GetCopertinaPortType'>
  <soap:binding style='rpc' transport=
              'http://schemas.xmlsoap.org/soap/http'/>
  <operation name='getCopertina'>
    <soap:operation soapAction=
        'urn:xmethods-delayed-quotes#getCopertina'/>
    <input>
      <soap:body use='encoded' namespace=
        'urn:xmethods-delayed-quotes' encodingStyle=
        'http://schemas.xmlsoap.org/soap/encoding/'/>
    </input>
    <output>
      <soap:body use='encoded' namespace=
        'urn:xmethods-delayed-quotes' encodingStyle=
        'http://schemas.xmlsoap.org/soap/encoding/'/>
    </output>
  </operation>
</binding>

<service name='GetCopertinaService'>
  <port name='GetCopertinaPort' binding=
                                  'GetCopertinaBinding'>
    <soap:address location='http://localhost
                  /ioProgrammo/soap/dummysoap.php'/>
  </port>
</service>
</definitions>
```

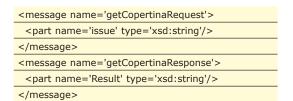
Il codice relativo al file WSDL non è così semplice come tutto il resto, tuttavia possiamo dare qualche indicazione di massima per renderlo maggiormente comprensibile.
Proviamo a recuperare le funzioni esposte tramite il solito *getfunction*.
Questa volta il risultato è

```
Array
(
    [0] => string getCopertina(string $issue)
)
```

il metodo copertina è definito in

```
<portType name='GetCopertinaPortType'>
  <operation name='getCopertina'>
    <input message='tns:getCopertinaRequest'/>
    <output message='tns:getCopertinaResponse'/>
  </operation>
</portType>
```

come si vede qui si dice che l'input dovrà essere di tipo *getCopertinaRequest* e l'output di tipo *getCopertinaResponse*. A loro volta i messaggi di input e output sono definiti in

```
<message name='getCopertinaRequest'>
  <part name='issue' type='xsd:string'/>
</message>
<message name='getCopertinaResponse'>
  <part name='Result' type='xsd:string'/>
</message>
```

Il resto del file è relativo alle dichiarazioni del formato WSDL ed alcune altre impostazioni che servono al buon funzionamento del tutto. Questi pochi dati non pretendono certamente di fornire una spiegazione approfondita del formato WSDL ma sono più che sufficienti per iniziare.
Seguendo questo template non dovreste comunque avere problemi nella creazione dei vostri servizi web.

USARE IL WEB SERVICE

Il client che usa questo Web Service è il seguente:

```
<?

   $client = new SoapClient('http://localhost
         /ioProgrammo/soap/dummysoap.php?WSDL');
   $issue = "89";
    print_r($client->getCopertina($issue));
?>
```

Che ci ritorna esattamente il titolo della copertina del numero 89 di ioProgrammo.

CONCLUSIONI

La teoria dei Web Service è piuttosto articolata e complessa. In questo articolo non volevamo addentrarci nei dettagli del protocollo SOAP o nei meandri delle definizioni sui WS. Volevamo semplicemente fornirvi un tutorial minimo e rapido per iniziare soprattutto ad usare Web Service presenti sul web e in parte anche per creare dei vostri servizi.
La creazione del file WSDL merita un approfondimento maggiore per potere essere ritenuta completa e sicuramente ne riparleremo.
Nel frattempo, utilizzando le informazioni contenute in questo articolo, dovreste essere già in grado di compiere i primi passi nel complesso mondo dei Web Service.

UN PO' DI RIGORE

**I Web Service esportano le proprie funzionalità tramite un protocollo predefinito: SOAP.
Tutti i linguaggi di programmazione ad alto livello hanno già al loro interno primitive tali da poter usare il protocollo usato dai Web Service, le comunicazioni fra client e server avvengono su TCP/IP, i messaggi sono sicuri perché incapsulati in formato SOAP.**

Una super cache per i nostri dati

Aggiriamo i colli di bottiglia imposti dai limiti d'accesso ai database, implementando nelle nostre applicazioni Java un meccanismo di caching dei dati e sfruttando una libreria OpenSource altamente affidabile

Una cache è un'area tipicamente locale alla macchina che esegue un'applicazione in una cache vengono memorizzate copie di dati acceduti più frequentemente, provenienti da una sorgente con tempi medi di accesso così elevati da rappresentare un collo di bottiglia per l'applicazione. Il caso più tipico è quello di un sito web che prende i dati da un database remoto. Chiaramente l'accesso sarebbe più veloce se i dati fossero prelevati da una directory locale alla macchina - ovvero una cache - una copia statica del database remoto. In questo modo è possibile incrementare le prestazioni riferendosi alla copia locale dei dati contenuti nella cache. Esempi di cache si trovano in molti dispositivi hardware e sistemi software: nei processori per migliorare l'accesso ai dati, in RAM, nei controller delle unità disco per migliorare i trasferimenti di file, nelle applicazioni server che costituiscono l'ossatura di Internet su rete, come sui DNS, per limitare il traffico generato da richieste di risoluzione di nomi di dominio. Ovviamente se da un lato lavorare su copie locali dei dati può incrementare le prestazioni, dall'altro nascono nuovi problemi, principalmente legati al possibile disallineamento tra i dati in cache ed i dati reali. Ad esempio, nel caso si decida di scrivere una nuova versione di un dato contenuto anche in una cache è necessario, per mantenere l'allineamento tra il dato originale e la copia, annullare quello in cache e scrivere sulla sorgente dati. Anche la lettura dei dati in presenza di cache risulta più macchinosa. Si tenta prima di tutto di accedere al dato nella cache. Qualora il dato non venga trovato, e si sia verificato quindi un cosiddetto *"cache miss"*, si legge il dato reale dalla sorgente, avendo l'avvertenza di copiare il dato letto anche nella cache in modo da averlo già a disposizione per una successiva lettura. Una cache ha tipicamente prestazioni superiori in termini di latenza e throughput rispetto alla sorgente dati reale ma una dimensione inferiore. Così l'altro problema principale nell'utilizzo di cache è come scegliere gli elementi da eliminare per far posto ai nuovi di cui è richiesta la memorizzazione. Alcune cache scartano gli elementi utilizzati meno frequentemente, altri quelli non utilizzati da più tempo. In questo articolo utilizzeremo *Ehcache*, una libreria Java open source che permette di utilizzare una cache nei propri applicativi e che ci svincola dall'implementare un nostro meccanismo di cache affidando invece la gestione ad *Ehcache* che è un progetto stabile e performante, utilizzato anche in Hibernate. Prepariamo l'area di lavoro creando la cartella *"ehcache"*, con le sottocartelle *"src"* dove troveranno posto i sorgenti Java, *"build"* dove saranno generati i file *.class*, *"lib"* dove saranno inserite le librerie necessarie e *"config"* per i file di configurazione e andiamo ad iniziare.

COME INIZIARE

Colleghiamoci al sito http://sourceforge.net/projects/ehcache e dalla sezione *"file"* scarichiamo l'ultima versione della libreria che al momento di scrivere l'articolo è la 1.1. Scompattiamo l'archivio e copiamo nella directory *"lib"* il file *"ehcache-1.1 .jar"*. Serve poi la libreria *commons-logging* prelevabile dall'indirizzo http://jakarta.apache.org/site/binindex .cgi#commons-logging. In questo caso il file da copiare nella directory *lib* è *"commons-logging-api.jar"*. Nel caso si utilizzi JDK1.2, o JDK1.3 serve anche la libreria *Xerces* prelevabile da http://xml.apache.org/xerces2-j /download.cgi. Ovviamente trovate tutto anche nel cd allegato alla rivista.

SCENARIO

Si supponga di dover sviluppare un sistema per la vendita online di biglietti di autobus che operano su tragitti autostradali. Il web server che si occuperà della vendita online è locato presso una server farm ma il database gira su una mac-

china negli uffici della sede. In una prima implementazione ogni richiesta di disponibilità di posti verrà rigirata al database. Si vedrà poi come l'uso di una cache possa migliorare decisamente le prestazioni del sistema. Scriviamo la classe *Bus* che ha la responsabilità di tenere traccia dei posti disponibili su ogni pullman. Ogni *Pullman* è caratterizzato da un codice identificativo, una capacità massima ed un numero di posti correntemente prenotato. Ovviamente la capacità massima non dovrà essere mai superata.

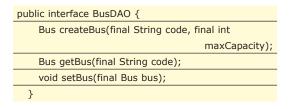
```
public final class Bus{
  private final String code;
  private final int maxCapacity;
  private int currentCapacity;
```

I metodi principali saranno *checkAvailability()* che restituisce il numero di posti ancora disponibili sul bus, e *book()* che prenota, previo controllo sulla disponibilità, il numero di posti specificato. Questi metodi sono dichiarati *syncrhonized*, in modo da evitare che due thread concorrenti possano prenotare gli stessi posti con il risultato che di vendere su un pullman più posti di quelli disponibili. A fronte di una prenotazione, il metodo *book()* utilizza un *BusDAO* di cui parleremo appena oltre, per salvare il proprio stato su database.

```
  synchronized public boolean checkAvailability(
                                        int seats){
    if(currentCapacity + seats <= maxCapacity)
                        {return true;}return false;}
  synchronized public boolean book(int seats){
    if (checkAvailability(seats)){
      currentCapacity = currentCapacity + seats;
      busDao.setBus(this); return true;
    }else{return false;}
}
```

ACCESSO AL DATABASE SENZA CACHE

Implementando il pattern *Data Access Object* sviluppiamo l'interfaccia *BusDAO* che espone le responsabilità di gestione della persistenza su database delle istanze della classe *Bus*, creandole ed aggiornando gli opportuni record su database. L'interfaccia *BusDAO* mette in evidenza i metodi *createBus()* tramite il quale l'applicativo può richiedere una nuova istanza di *Bus*, specificandone codice identificativo e capacità massima, *getBus()* che restituisce l'istanza di *Bus* corrispondente ad un definito codice, *setBus()* che permette aggiornare il record di un bus passato come parametro. Specifichiamo un'interfaccia cosicché si nascondano i dettagli implementativi della classe *DAO*. Questo ci consentirà di implementare due diverse classi una che esegue la gestione dei database senza utilizzare encache, la seconda che invece ottimizza le prestazioni tramite encache. In questo modo sarà semplice effettuare in un secondo momento il confronto fra le prestazioni

```
public interface BusDAO {
    Bus createBus(final String code, final int
                                     maxCapacity);
    Bus getBus(final String code);
    void setBus(final Bus bus);
  }
```

Inizialmente scriveremo l'implementazione che accede direttamente al database senza fare uso della cache. Il metodo *createBus()* eseguirà un *"insert"* per creare un record relativo al nuovo bus ed invocherà *setDao()* sulla nuova istanza per impostare il *BusDAO* utilizzato nel metodo *book()*, il metodo *getBus()* eseguirà un *"select"* per caricare i valori degli attributi del pullman richiesto, il metodo *setBus()* eseguirà un "update" per aggiornare i valori dei campi che rappresentano lo stato del bus passato come parametro.

```
  public class BusDAODirect implements BusDAO {
    public Bus createBus(final String code, final int
                                     maxCapacity){
      bus = new Bus(code, maxCapacity);
      bus.setDao(this);
      //SQL INSERT
      return bus;}
    public Bus getBus(final String code){ //... }
    public Bus setBus(final Bus bus){ //... } }
```

UN'APPLICAZIONE DI TEST

Sviluppiamo una classe che utilizzi le classi *Bus* e *BusDAO*. Un esempio potrebbe essere quello di creare un ristretto numero di istanze di bus e provare ad invocarne i metodi di richiesta di disponibilità e prenotazione di posti. Qui di seguito una traccia.

```
public class CachedApplication {
  private static final BusDAO busDAO =
                              new BusDAODirect();
  public static void main(String[] a){
                     new CachedApplication().run();}
  public static BusDAO getBusDAO(){return busDAO;}
  public void run(){
    Bus bus = busDAO.createBus("bus001", 60);
    bus.book(3);} }
```

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

È importante notare come la classe principale esponga tramite *getBusDAO()* un'istanza di *BusDAO* utilizzabile dalla classe *Bus*.

COMPILAZIONE E RISULTATI

Compiliamo l'applicazione portandoci nella directory *"src"*, invocando *javac*

```
javac -classpath ..\lib\commons-collections-3.1.jar;..
    \lib\commons-logging-api.jar;..\lib\ehcache-1.1.jar
                                    -d ..\build *.java
```

Lanciamo poi l'applicazione spostandoci nella cartella *"build"* e digitando

```
java -cp ..\lib\commons-collections-3.1.jar;..
                  \lib\commons-logging-api.jar;..\lib
        \ehcache-1.1.jar;..\config;.\ CachedApplication
```

Se per ogni sessione utente viene ottenuta un'istanza dal database per controllarne la disponibilità. È semplice immaginare come questo si ripercuota direttamente nell'esecuzione di una *"select"* su database remoto, creando del traffico inutile poiché il bus in esame è stato magari appena consultato da un un altro utente, e quindi potenzialmente potrebbe già trovarsi in memoria. Facciamo girare l'applicativo e misuriamo il tempo speso nell'attesa di una risposta da parte del database server remoto.

tempo di accesso al db in millisecondi: 5317
insert eseguiti:4, update eseguiti:48,
selects eseguiti:400

FACCIAMOLO CON EHCACHE

Ehcache permette la configurazione della cache programmaticamente o dichiarativamente con l'utilizzo di file XML. Ecco la configurazione utilizzata per questo esempio, da copiare nella cartella *"config"* con nome *"buscache.xml"*.

```
<ehcache>
   <diskStore path="java.io.tmpdir"/>
   <defaultCache
   maxElementsInMemory="10000" eternal="false"
      timeToIdleSeconds="120" timeToLiveSeconds="120"
      overflowToDisk="false" diskPersistent="false"
      diskExpiryThreadIntervalSeconds="120"/>
   <cache name="bus"
      maxElementsInMemory="10000" eternal="false"
      overflowToDisk="false" timeToIdleSeconds="300"
      timeToLiveSeconds="600"/>
</ehcache>
```

L'elemento *"diskstore"* indica dove devono essere memorizzati gli elementi in cache. È possibile specificare un path completo oppure una variabile d'ambiente Java. In questo caso *"java.io-.tmpdir"* punta alla directory temporanea.
L'elemento *"defaultcache"* imposta la configurazioni di default per tutte le cache. Sarà poi necessario specificare solo i parametri che differiscono dal default.
Ecco il significato degli attributi:

- **Name:** nome della cache. Dal programma si utilizzerà tale nome per avere un puntatore alla cache.
- **Eternal:** se *false* gli elementi nella cache vengono comunque eliminati dopo un periodo configurabile durante il quale non sono stati acceduti, se *true* gli elementi non sono mail eliminati dopo un timeout ma solo per far posto a nuovi elementi.
- **MaxElementsInMemory:** numero massimo di elementi nella cache in RAM.
- **OverflowToDisk:** se *true,* tutti gli elemeni rimossi dalla cache in ram sono scritti mediante serializzazione in una cache su disco. Se *false* vengono invece semplicemente rimossi.
- **TimeToIdleSeconds:** se un elemento in cache non viene utilizzato per un periodo maggiore al valore di questo attributo, viene scartato dalla cache.
- **TimeToLiveSeconds:** un elemento creato nella cache permane per un periodi pari al valore di questo attributo, dopodiché viene rimosso.
- **DiskPestisten:** se *true,* la cache su disco viene ricaricata ad un nuovo riavvio della cache, altrimenti ad ogni avvio, la cache su disco risulterà vuota.
- **DiskExpiryThreadIntervalSeconds:** ogni quanti secondi viene eseguito un controllo sulla validità degli elementi memorizzati nella cache su disco.

ACCESSO AL DB CON CACHE

Scriviamo una nuova implementazione di *BusDAO* che aggiunge caratteristiche di caching al *DirectBusDAO.* Strutturalmente la nuova classe si frapporrà tra l'applicazione e il *DirectBusDAO* che accede alla sorgente dati, gestendo la cache e richiamando i metodi del *DirectBusDAO* solo quando servono.

NOTA

ALGORITMI DI CACHING

Molti sono gli algoritmi di caching utilizzati.Tutti indicano una politica da utilizzare per decidere quali elementi nella cache rimuovere a fronte della necessità di inserire un nuovo elemento:

- **Least frequently used (LFU): i candidati all'eliminazione sono tutti quelli che hanno un rapporto tra numero di accessi e tempo di permanenza nella cache basso.**
- **Least recently used (LRU): viene eliminato l'oggetto che non è acceduto da maggior tempo.**
- **First in first out (fifo): è come se la cache fosse un tubo, da un capo gli elementi sono messi in cache, dall'altro ne fuoriescono. Quando si rende necessario inserire un nuovo elemento questo semplicemente “spinge” fuori dalla cache quello in ultima posizione.**

Il fatto che entrambe le classi *DAO* estendano la stessa interfaccia fa sì che il client le possa utilizzare indistintamente senza che il codice debba essere cambiato.

```
import net.sf.ehcache.*;
public class CachedBusDAO implements BusDAO {
  private final BusDAO source;
  private static final CacheManager cacheManager;
  private final Cache cache;
  static {
    try{cacheManager = CacheManager.create(
                                      "build/buscache.xml");
    }catch(CacheException ce){throw new
                                   RuntimeException(ce);} }
  public BusDAOCached(final BusDAO source){
    this.source = source;
    cache = cacheManager.getCache("bus"); }
  public Bus createBus(String code, int maxCapacity) {
    Bus bus = source.createBus(code, maxCapacity);
    return bus; }
  public Bus getBus(String code) {
    //Element rappresenta qualsiasi oggetto in una
                                           cache Ehcache
    Element e = null; Bus bus = null;
    try{
      //Ricerca di un Element nella cache associato al
                                                  codice
      e = cache.get(code);}
    catch(Exception ex){throw new
                                    RuntimeException(ex);}
    if(e == null){
      bus = source.getBus(code);
      //Inserimento del bus in cache sottoforma di
                              Element associato al codice
      cache.put(new Element(code, bus));
    }else{bus = (Bus)e.getValue();}
      return bus;}
  public void setBus(Bus bus) {
    cache.remove(bus.getCode());
    source.setBus(bus);}
}
```

Nel costruttore viene passato un *BusDAO* a cui verrà demandato il compito di interagire effettivamente con il database. Nell'inizializzatore statico viene creato un *CacheManager* partendo dal file di configurazione scritto nei passi precedenti. Il metodo *createBus()* è una semplice delega. Il metodo *getBus()* cerca l'istanza in cache. In caso positivo restituitce l'istanza trovata risparmiando un accesso al database, in caso negativo delega al *busDAO* sorgente la ricerca su database, posizionando l'istanza trovata in cache per velocizzare probabili futuri accessi. Il metodo *setBus()* elimina prima di tutto la copia in cache poiché disallineata rispetto al nuovo stato e delega al *busDao* sorgente il compito di aggiornare il record su database.

UTILIZZARE LA VERSIONE CON CACHE

È necessaria una piccola modifica nelle classi per abilitare l'utilizzo del *busDAO* con la versione dotata di caching. La prima modifica è nell'applicazione vera e propria.

```
private static final BusDAO busDAO = new
                      BusDAOCached(new BusDAORaw());
```

La seconda è nella classe *bus* che deve essere dichiarata Serializable per far si che istanze di *Bus* in cache RAM possano essere scritte su disco.

```
import java.io.Serializable;
public final class Bus implements Serializable{
```

Compiliamo l'applicazione portandoci nella directory "src", invocando javac

```
javac -classpath ..\lib\commons-collections-3.1.jar;..
    \lib\commons-logging-api.jar;..\lib\ehcache-1.0.jar
                                        -d ..\build *.java
```

Lanciamo poi l'applicazione spostandoci nella cartella "build" e digitando

```
java -cp ..\lib\commons-collections-3.1.jar;..
   \lib\commons-logging-api.jar;..\lib\ehcache-1.0.jar;
                          ..\config;.\ CachedApplication
```

Compiliamo e facciamo girare l'applicativo.
I risultati ottenuto sono notevolmente migliori.

tempo di accesso al db in ms: 1041
insert:4, update:42, selects:0, hits:357, misses:43

tting Started Latest Headlines

Benchmark	EHCache	JCS	Difference
insert 5 million typical CacheElements. Then get each of them by key.	22.704s	42.176s	79% faster
insert 5,000,000 typical CacheElements. Then remove them one at a time by key	14.999s	44.536s	293% faster

Contributions

wotif.com

The ehcache project gratefully acknowledges the contribution of wotif.com, who granted a license to the authors to in respect of code developed at wotif.com for the purpose of making it available to the community under an Apache open source license.

Thanks also to:

- ThoughtWorks for free legal assistance in formalising the granting of code from wotif.com to the open source community. Copies of the executed deeds are available should anyone need to audit the grant.
- Cenqua for the free open source license of the test coverage tool Clover.
- Atlassian for the free open source licenses of JIRA, an issue tracking tool and Confluence, a wiki.

Finally, thanks to those who have contributed to the project. Those people are acknowledged in the traditional way with an @author tag in the source code.

About the ehcache Name and Logo

ehcache is a palindrome: it reads the same left to right, and right to left. The logo is similarly symmetrical, and is evocative of the diagram symbol for a doubly-linked list. The JDK1.4 LinkedHashMap, and Apache's LRUMap are a HashMap with a doubly-linked list running through all of its entries. These structures lie at the heart of ehcache.

SOURCEFORGE.net Support this project

Fig. 1: *Alcuni test presenti direttamente sul sito di encache mostrano un miglioramento di oltre il 70% rispetto a JCS*

CONCLUSIONI

L'esempio ha dimostrato quanto sia semplice aggiungere un livello di cache ad una classe che accede ad una sorgente dati, in modo da aumentarne drammaticamente le prestazioni.

Daniele De Michelis

CACHE HIT E CACHE MISS

Nell'utilizzo di cache ci sono due eventi significativi: il cache hit avviene quando si cerca un elemento nella cache e lo si trova con successo, l'evento opposto è il cache miss, quando l'elemento non si trova e diventa necessario accedere o ad una cache di livello superiore o direttamente alla data source.

Firefox, il browser estensibile

Una guida passo passo per creare un'estensione per Firefox utilizzando XUL il linguaggio semplice e veloce derivato da XML e utilizzato per interfacce grafiche di grande effetto

Xul è un linguaggio basato su XML creato dalla Mozilla Foundation per scrivere facilmente e velocemente interfacce grafiche. Proprio XUL è stato scelto anche per la realizzazione di estensioni per il browser Firefox. La dove per estensione si intende un piccolo programma che "estende" e "personalizza" il Browser adattandolo alle vostre esigenze. Voi stessi potete creare un'estensione per il vostro browser. Tutto quello di cui avete bisogno è un normalissimo editor di testo e un compressore *.zip*. Se poi siete dei professionisti della programmazione potete anche utilizzare un editor XML per editare il vostro file *.xul*.
Come avrete già capito il cuore di un estensione per Firefox è costituito proprio da un file *.xul*, in questo articolo vi guideremo alla creazione di una miniestensione.

ANATOMIA DI UN FILE .XUL

Innanzitutto un file *".xul"* è effettivamente un file xml e inizia con le classiche dichiarazioni:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/"
                                   type="text/css"?>
```

A seguire, solitamente troviamo la dichiarazione del documento: una "window" con la specifica del namespace a cui appartiene:

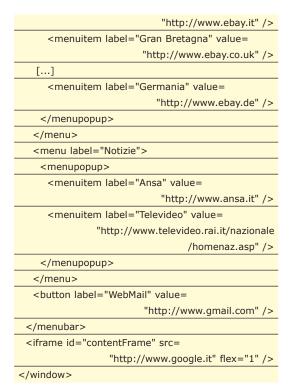
```
<window xmlns="http://www.mozilla.org
          /keymaster/gatekeeper/there.is.only.xul"
          orient="vertical" title="Bookmarks">
```

All'interno della <*window*> possiamo aggiungere tutti i componenti grafici di cui la nostra applicazione necessita e che possono essere nativi di xul, o importati dall'html:

```
<menupopup id="menu_ToolsPopup">...
                                   </menupopup>
<image src="images/banner.jpg"/>
<html:button xul:onclick="DoSomething()">
```

Così facendo riusciamo, abbastanza facilmente e velocemente, a comporre un'interfaccia grafica. In un articolo pubblicato nel numero 91 di ioProgrammo, per esempio, avevamo costruito una barra in cui inserire dei bottoni per accedere alle diverse sezioni di un sito, o per organizzare meglio i nostri bookmark:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/"
                                   type="text/css"?>
<window xmlns="http://www.mozilla.org
          /keymaster/gatekeeper/there.is.only.xul"
          orient="vertical" title="Bookmarks">
  <script type="application/x-javascript">
   function loadURL(event) {
     var contentFrame = document.getElementById(
                                   'contentFrame');
     var url = event.target.getAttribute('value');
     if (url) contentFrame.setAttribute('src', url);
   }
  </script>
  <menubar oncommand="loadURL(event);">
    <menu label="Programmazione">
      <menupopup>
        <menuitem label="IoProgrammo" value=
                   "http://www.ioprogrammo.it/" />
        <menuitem label="Javastaff"
                   value="http://www.javastaff.com/" />
        <menuitem label="XulPlanet"
            [ ...]
      </menupopup>
    </menu>
    <menu label="Aste">
      <menupopup>
        <menuitem label="Italia" value=
```

```
                    "http://www.ebay.it" />
    <menuitem label="Gran Bretagna" value=
                    "http://www.ebay.co.uk" />
  [...]
    <menuitem label="Germania" value=
                    "http://www.ebay.de" />
   </menupopup>
  </menu>
  <menu label="Notizie">
   <menupopup>
    <menuitem label="Ansa" value=
                    "http://www.ansa.it" />
    <menuitem label="Televideo" value=
          "http://www.televideo.rai.it/nazionale
                    /homenaz.asp" />
   </menupopup>
  </menu>
  <button label="WebMail" value=
                    "http://www.gmail.com" />
 </menubar>
 <iframe id="contentFrame" src=
          "http://www.google.it" flex="1" />
</window>
```

Se rendiamo disponibile questo file xul in un Web Server, gli utenti che lo apriranno con Firefox, vedranno una barra di navigazione con dei menu a tendina che ci permettono di scegliere velocemente la pagina nel frame sottostante. Il controllo degli eventi è affidato alla funzione javascript *loadURL()* incapsulata nei tag *<script>* e *</script>*.

LA STRUTTURA DELLE DIRECTORY

Abbiamo capito ormai che tipo di struttura ha un'interfaccia scritta appositamente per Firefox. Si progetta la parte grafica utilizzando la sintassi di xul, mentre il controllo delle interazioni è assegnato a funzioni in un linguaggio di scripting supportato dal browser, solitamente javascript. Se vogliamo scrivere un'estensione vera e propria da installare nel nostro browser dovremo solo procurarci un compressore zip, dato che i file *".xpi"* ovvero i file autoinstallanti per le estensioni Mozilla altro non sono che archivi in quel formato, poi rinominati e creare l'archivio seguendo qualche accorgimento.
Per prima cosa occorre disporre i file che compongono la nostra estensione in cartelle opportunamente scelte. Iniziamo con il replicare la seguente struttura

esempio_jar
esempio_jar\content
esempio_jar\content\esempio
esempio_jar\skin
esempio_jar\skin\classic\esempio
esempio_xpi
esempio_xpi\chrome

FILES STANDARD

A questo punto dobbiamo riempire le varie cartelle. Alcune conterranno immagini e suoni per far funzionare la nostra applicazione, altre conterranno dei file di descrizione che serveranno all'estensione per autoinstallarsi correttamente. Iniziamo creando un file "descrittore" chiamato *contents .rdf* nella cartella *esempio_jar\content\esempio*.

TESTARE LE ESTENSIONI

Se siete in fase di sviluppo di una nuova estensione è utile installare una seconda instanza di Firefox per testare il vostro lavoro.
Infatti un qualunque problema provocato da una non corretta generazione del file .xpi potrebbe provocare il mancato riavvia di Firefox dopo avere installato la nuova estensione.
In tal caso dovreste procedere a un riavvio in safe mode.
Molto più comodo "sporcare" un'installazione di Firefox dedicata esclusivamente ai test.

Questo file, che si occupa di specificare tutte le risorse contenute nell'estensione.

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org
                    /1999/02/22-rdf-syntax-ns#"
xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
<RDF:Seq RDF:about="urn:mozilla:package:root">
    <RDF:li RDF:resource=
                    "urn:mozilla:package:esempio"/>
</RDF:Seq>
<RDF:Seq RDF:about="urn:mozilla:overlays">
<RDF:li RDF:resource="chrome://browser
                    /content/browser.xul"/>
<RDF:li RDF:resource="chrome://navigator
                    /content/navigator.xul"/>
</RDF:Seq>
<RDF:Seq RDF:about="chrome://browser
                    /content/browser.xul">
<RDF:li>chrome://esempio/content/esempioOverlay
                    .xul</RDF:li>
</RDF:Seq>
<RDF:Seq about="chrome://navigator/content
                    /navigator.xul">
<RDF:li>chrome://esempio/content/esempioOverlay
                    .xul</RDF:li>
</RDF:Seq>
<RDF:Description RDF:about=
                    "urn:mozilla:package:esempio"
chrome:displayName="Ciao da IoProgrammo."
chrome:author="Luca Mattei"
chrome:authorURL="mailto:luca.mattei@javastaff.com"
```

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

```
chrome:name="esempio"
chrome:extension="true"
chrome:description="La nostra prima estensione.">
</RDF:Description>
</RDF:RDF>
```

Il File comincia con la consueta dichiarazione della versione xml e del namespace a cui appartiene il nostro documento. Seguono un elenco di dichiarazioni di risorse, indicate con il tipo di indirizzamento *chrome://*.
Per esempio consideriamo questo indirizzo:

```
chrome://browser/content/browser.xul
```

Si tratta di un riferimento al frame interno di Firefox. L'indirizzo segue, nella composizione, delle semplici regole. La prima parola dopo il prefisso *chrome://* rappresenta il package a cui le risorse, divise in directory, appartengono. *"browser"* è il package delle risorse interne a Firefox, mentre *"navigator"* serve per mantenere una compatibilità alle vecchie versioni di Mozilla. Tutte le nostre risorse sono accessibili quindi a partire dall'indirizzo:

```
chrome://esempio/
```

NOTA

CHE COSA È RDF?
contents.rdf **è un file** ***RDF (Resource Description Framework)*** **che serve a descrivere i contenuti di una estensione. RDF è una grammatica XML che fornisce un modello di dati facilmente processabile da una applicazione. Ulteriori informazioni su questo formato possono essere sul sito del W3C:**
http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

L'ESTENSIONE

Nel file *contents.rdf* l'unico riferimento ad un file della nostra estensione è:

```
chrome://esempio/content/esempioOverlay.xul
```

Questo file, si occupa di disaccoppiare le funzionalità dell'estensione dal file xul vero e proprio.
Nel nostro esempio abbiamo definito una sola funzione:

```
<?xml version="1.0"?>
<overlay id="esempioOverlay" xmlns=
        "http://www.mozilla.org/keymaster/gatekeeper
                                /there.is.only.xul">
<script type="application/x-javascript" src=
    "chrome://esempio/content/esempioOverlay.js" />
<menupopup id="menu_ToolsPopup">
<menuitem insertafter="devToolsSeparator" label=
                   "Esempio IoProgrammo" accesskey=
                       "e" oncommand="esempio();" />
</menupopup>
</overlay>
```

Viene definito infatti un menu popup, inserito nella sezione *Tools* o *Strumenti* della barra menu di Firefox, con un unico *menuitem,* che esegue la funzione javascript *esempio().*

Tutte le funzioni javascript vengono inserite nel file *esempioOverlay.js,* come indicato nel file appena visto. Il nostro *esempioOverlay.js* conterrà ovviamente solo una alert che visualizzi il classico *"Hello World"*:

```
function esempio()
{
        alert("Ciao Mondo!");
}
```

Questi due file costituiscono la parte più dinamica dell'estensione, quella che intendiamo modificare fino a raggiungere il comportamento funzionale desiderato. Disaccoppiando questa parte dai file xul, abbiamo anche un altro vantaggio, poter utilizzare gli stessi overlays in più file xul, limitandoci ad importarli ogni volta:

```
<?xul-overlay href="chrome://esempio/content
                        /esempioOverlay.xul"?>
```

Da questo indirizzo chrome, è facile intuire che andremo a salvare i file di *Overlay* nella cartella - *esempio_jar\content*

ADATTARE L'ESTENSIONE ALLO SKIN

Avrete sicuramente notato che fra le cartelle da creare c'è: - *esempio_jar\skin*, questo perché in un'estensione è possibile creare diversi profili grafici, che il browser caricherà a seconda della skin selezionata nelle impostazioni generali. Nel nostro *HelloWorld* non abbiamo alcun bisogno di creare diverse skin, per questo ci limiteremo al minimo indispensabile, la skin predefinita: *"classic"*.
Per questo, creiamo il file *esempio_jar\skin\classic\esempio\contents.rdf.*
Possiamo tranquillamente utilizzare il template classico per estensioni senza particolari profili grafici:

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF=
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
  <RDF:Seq about="urn:mozilla:skin:root">
    <RDF:li resource="urn:mozilla:skin:classic/1.0" />
  </RDF:Seq>
  <RDF:Description about="urn:mozilla:skin:classic/1.0">
    <chrome:packages>
      <RDF:Seq about="urn:mozilla:skin:classic
                              /1.0:packages">
        <RDF:li resource="urn:mozilla:skin:classic
                              /1.0:esempio" />
      </RDF:Seq>
```

```
    </chrome:packages>
  </RDF:Description>
</RDF:RDF>
```

Eventuali file da utilizzare nella particolare skin vanno inclusi nella cartella e richiamati con la sintassi:

```
chrome://esempio/skin/nomefile.png
```

ULTIMI PASSI

Arrivati a questo punto, abbiamo creato tutti i file che devono essere inclusi nella cartella *esempio_jar*. Tutti i file e le sottocartelle di questa directory vanno inclusi in un archivio zip, facendo attenzione che tutti i percorsi dei file inclusi siano relativi, come nell'immagine.

Fig. 1: ***L'archivio esempio.jar non è altro che un file zip rinominato. Nel crearlo facciamo particolare attenzione ai percorsi dei file***

Una volta creato l'archivio lo rinominiamo in *esempio.jar* e lo salviamo nella cartella *esempio_xpi\ chrome*. In pratica Firefox sa che ogni volta che deve accedere ad una risorsa passando per un indirizzo *chrome://esempio* deve andare ad attingere da questo archivio.
Passiamo alla cartella *esempio_xpi*, che oltre al jar appena creato, deve contenere due file: *install.rdf* e *install.js*. Il primo è fondamentale, perché serve a Firefox in fase di installazione, il secondo solo per retrocompatibilità con le versioni di Mozilla precedenti. Vediamo il primo:

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02
                              /22-rdf-syntax-ns#"
xmlns:em="http://www.mozilla.org/2004/em-rdf#">
<Description about="urn:mozilla:install-manifest">
<em:id>{ee9692e7-1e80-41de-88c8-09310124abde
                                      }</em:id>
<em:name>IoProgrammoEsempio</em:name>
<em:version>0.1</em:version>
<em:description>Semplice Helloworld
                              </em:description>
<em:creator>Luca Mattei</em:creator>
<em:homepageURL>http://www.ioprogrammo.it
                              </em:homepageURL>
<em:file>
<Description about="urn:mozilla:extension:
                              file:esempio.jar">
<em:package>content/esempio/</em:package>
<em:skin>skin/classic/esempio/</em:skin>
</Description>
</em:file>
<em:targetApplication>
<Description>
<em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384
                                      }</em:id>
<em:minVersion>0.7</em:minVersion>
<em:maxVersion>1.1</em:maxVersion>
</Description>
</em:targetApplication>
</Description>
</RDF>
```

Particolare attenzione va riservata agli id dell'applicazione target e dell'estensione.
Per quanto riguarda il primo, questo deve essere sempre lo stesso, dato che identifica in maniera univoca Firefox:

```
<em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384
                                      }</em:id>
```

L'id della nostra estensione invece deve essere univoco, per non far confusione con altre estensioni. Per averne uno tutto nostro, possiamo ricorrere a due vie, un programma standalone da far girare sul nostro PC, oppure una cgi in perl liberamente utilizzabile dal web a questo indirizzo: *http://extensions.roachfiend.com/cgi-bin/guid.pl*.
Il programma standalone esiste sia in versione windows *(GUIDGen)*, liberamente scaricabile dal sito Microsoft, sia in versione Linux, utilizzabile da shell richiamando *uuidgen*.
Install.js è l'equivalente in javascript di *install.rdf* e potremo ometterlo tranquillamente, se non intendiamo rendere compatibile la nostra estensione con i vecchi Mozilla.

IL FILE XPI

Ora l'estensione è pronta, costruiamo un archivio zip a partire dalla cartella *esempio_xpi*, stando attenti ai path dei singoli file, come avevamo fatto per il file *jar*. Rinominiamo il file in *esempio.xpi*, formato scelto da Mozilla per contraddistinguere i suoi plug-ins e siamo pronti per l'installazione.
Basta andare nel menu del nostro amato Browser, cliccare su *"Apri file.."* e scegliere la nostra nuova creazione. Se tutto va bene, riusciremo ad installare il nostro *HelloWorld* e potremo eseguirlo dal menu strumenti.

Luca Mattei

Sono disponibile per chiarimenti, critiche e suggerimenti all'indirizzo:
luca.mattei@javastaff.com

Un File System portatile in Java

Utilizzando una libreria OpenSource di Jakarta costruiremo un'applicazione che utilizza internet come un enorme disco rigido virtuale. Inoltre diremo qualcosa sui server Web e i database...

Il nostro problema è il seguente: come moltissime persone abbiamo tanti file sparsi su internet. Qualcosa lo abbiamo lasciato su un FTP, qualcosa in ufficio in una directory condivisa, altra roba l'abbiamo sparsa sul file system di Gmail o su altri file system non locali. Il punto è che andiamo spesso in giro, perciò sarebbe utile mantenere una copia locale di tutti i file che abbiamo sparso sincronizzandoli con il nostro Hard Disk. È anche vero che andando spesso in giro non usiamo sempre lo stesso portatile, o la stessa periferica mobile. Perciò lo scopo di questo articolo è il seguente:

1) Realizzare un client Java scaricabile da internet.
2) Utilizzare il client per autenticarsi su un server.
3) Sulla base dell'autenticazione eseguire la sincronia con le nostre risorse remote e il file system locale su cui stiamo lavorando.

I vantaggi di questo sistema stanno nel fatto che, l'unico file che è necessario avere a disposizione per eseguire la sincronia è il client, e possiamo scaricarlo da internet. La lista dei file da sincronizzare risiederà anche essa sul server perciò realmente oltre che del client non abbiamo bisogno di niente.
Una volta autenticato eseguiamo la sincronizzazione e ci ritroveremo sulla macchina locale i file che mi servono.

COME FARE?

Per i nostri scopi utilizzeremo Common VFS di Jakarta. Questo è un progetto opensource con il quale si vuole creare una libreria stabile per gestire diversi tipi di file e risorse (*ftp, http, smb* etc etc.) come una singola risorsa. In questo modo agli occhi dello sviluppatore Java la manipolazione di una risorsa o di un'altra è del tutto trasparente. Nella nostra applicazione l'utente comunica inizialmente con un server che tiene memorizzati i nomi di login, con la relativa password e la lista di risorse registrate dall'utente. In questo modo dopo una prima autenticazione abbiamo a disposizione la lista di tutte le nostre risorse sparse nella rete. Ora il client può tranquillamente scegliere una cartella sul computer dove si trova, da tenere sincronizzata, aggiungendo o rimuovendo file e cartelle.
In automatico il programma, in base alla lista delle nostre risorse remote, aggiorna i nostri repository, trasferendo o cancellando i file. C'è una certa "complicazione" nel mantenere su un server remoto una

Conoscenze richieste
J2SE, Servlet, JDBC

Software
J2SE SDK, Tomcat, Common VFS

Impegno
Tempo di realizzazione
COSA È JAKARTA

Jakarta è un progetto per lo sviluppo di codice *opensource*, riutilizzabile in diversi ambiti. All'interno del sito principale http://jakarta.apache.org troviamo una suddivisione tra i prodotti per tipologia: librerie, framework e server. Sfogliando tutti i progetti che sono ospitati da Jakarta possiamo trovare tantissimi software da utilizzare nelle nostre applicazioni evitando così, ogni volta, di reinventare l'acqua calda. All'interno di questa lunga lista di librerie e tool troviamo una sezione che può essere utile conoscere in molti progetti, la sezione "Commons". Questa racchiude tutti i progetti dedicati alla riusabilità di componenti Java. All'interno di questa sezione troviamo due ulteriori sottosezioni: Proper, repository di componenti Java riutilizzabili e Sandbox, workspace che non prevede la durata e la manutenzione dei progetti dei partecipanti. Proprio all'interno di Sandbox troviamo il componente molto interessante che utilizzeremo nel nostro progetto: *Common VFS (Virtual File System).*

lista di utenti con le relative risorse associate, ma se in un futuro volessimo vendere il nostro client il sistema ci verrebbe molto utile.

TRASFERIMENTO FILE

L'interfaccia *FileSystemManager*, inclusa nel package *org.apache.common.vfs*, ci permette di gestire i file system a nostra disposizione. Ecco un semplice modo per connetterci ad un file system

```
FileSystemManager fsManager= VFS.getManager();
FileObject aFile = fsManager.resolveFile(
                        "ftp://user:pass@server:21" );
```

In questo modo abbiamo inizializzato un *FileObject*, un'interfaccia che rappresenta un file. Un file può essere una directory o un semplice file. Avremo quindi a disposizione diversi metodi per elencare i file, crearli e distruggerli. Definiamo un metodo che ci servirà di sicuro, ovvero il metodo per trasferire file dal nostro computer ai computer sparsi sulla rete e viceversa scaricare i nostri file dalla rete.

```
public static void copia(String from,String to)
                                    throws Exception
{
  FileObject src = fsManager.resolveFile(from);
  FileObject dest = fsManager.resolveFile(to);
  if(dest.exists() && dest.getType() == FileType.FOLDER)
    dest = dest.resolveFile(src.getName()
                                    .getBaseName());
    dest.copyFrom(src, Selectors.SELECT_ALL);
}
```

Questo metodo richiede come parametri due indirizzi, sotto forma di stringa, i quali saranno rispettivamente la sorgente da cui copiare e la destinazione. Una volta risolto l'indirizzo del file in un oggetto *FileObject* possiamo tranquillamente copiare il file dalla sorgente alla destinazione passata nell'argomento. In questo modo, in maniera del tutto trasparente per il programmatore, avviene il trasferimento del file, rispettando alla perfezione il protocollo della risorsa remota.
Non ci resta che pensare a come organizzare l'applicazione che vogliamo sviluppare. All'inizio il nostro programma verrà inizializzato passando come argomento una cartella, all'interno della quale troveremo due diverse sottocartelle che rappresenteranno rispettivamente i file remoti scaricati e i file locali da inviare. Quindi dovremo preoccuparci di sincronizzare la cartella *Download* con tutti i nostri file sparsi sulla rete. Poi dovremo attivare un Thread che periodicamente controllerà la presenza di file nella cartella *Upload* e, nel caso ci fossero, inviarli verso i nostri server remoti.

SINCRONIZZAZIONE

Supponiamo di avere a nostra disposizione la lista dei vari computer su cui risiedono i nostri file. Per il momento immaginiamo di averla, già disponibile, successivamente vedremo come ottenerla.
Dunque, abbiamo a disposizione il nome della cartella locale dove mettere i file e la lista dei vari file system remoti e non rimane altro che ciclare su ogni file system e scaricare tutti i file usando la funzione che abbiamo definito in precedenza

```
public static void initialSync(Vector list) throws
                                          Exception{
  System.out.println("Sincronizzazione iniziale..");
    fslist=list;
    fsManager = VFS.getManager();
    for (int i=0;i<fslist.size();i++) {
      FileObject fo = fsManager.resolveFile((String)
                              fslist.elementAt(i));
      FileObject[] fofiles = fo.getChildren();
      for ( int j = 0; j < fofiles.length; j++ ){
        copia(fofiles[j].getName().getURI(),
                                      downloadDir);
        System.out.println(fofiles[j].getName()
                                  +" copiato");} }
  System.out.println("Sincronizzazione iniziale finita.");
}
```

Alla fine di questo metodo abbiamo sincronizzato tutti i nostri file nella cartella *downloadDir*. Ora dobbiamo preoccuparci dell'upload. Come già detto abbiamo a disposizione una cartella di transito, nella quale i file che devono essere trasferiti ai server remoti sostano per un tot di tempo. La soluzione in questo caso può essere di definire un classe che estende *TimerTask* e che controllerà periodicamente la cartella per vedere se ci sono file in transito.
In questa fase definiamo staticamente ogni quanti secondi verrà risvegliato il *TimerTask* per il controllo, poi magari potremmo farlo definire dall'utente. C'è un'ulteriore riflessione da compiere. Disponiamo della lista dei nostri server, ma su quale effettueremo l'upload? Potrebbero esserci mille vie e mille motivi per definire un ordine di preferenza nei server, in questo semplice programma effettueremo un round robin tra i vari file system remoti, scegliendo progressivamente rispetto alla lista di file system che abbiamo per effettuare l'upload. Dopo aver sincronizzato la nostra cartella locale facciamo partire il *TimerTask*

```
Timer    timer = new Timer();
UploadController uc=new UploadController(this);
timer.schedule( uc, 5000, 5000 );
```

Il nostro *TimerTask* verrà avviato ogni 5 secondi per controllare la presenza di nuovi file. Ecco quindi l'implementazione della classe *UploadController*

COMMON VFS

Common VFS richiede il download di molte librerie presenti sul sito di Jakarta. Ogni libreria riguarda un protocollo particolare, quindi gli sviluppatori di Common VFS hanno dovuto definire uno standard per le risorse a cui accedere. Nel momento in cui ci colleghiamo ad una risorsa specifica viene richiamata l'implementazione relativa, senza che noi la specifichiamo direttamente nel codice.

VFS

***(Virtual File System)* è una tecnologia che in molti vogliono sviluppare all'interno di sistemi operativi e programmi. È un'astrazione del file system e quindi porta benefici dal punto di vista dello spazio ma chiaramente si possono incontrare dei problemi per quanto riguarda l'implementazione dell'architettura. Nei seguenti link trovate dei progetti riguardanti VFS**

http://developer.gnome.org/doc/API/gnome-vfs/
http://www.cse.unsw.edu.au/~neilb/oss/linux-commentary/vfs.html
http://www.parl.clemson.edu/pvfs/desc.html

```
class UploadController extends TimerTask {
  Kernel kernel;
  String uploadDir;
  public UploadController(Kernel kernel) {
    //Passiamo come argomento il kernel in modo
    //tale che questa classe potrà accedere alle
    //informazioni e ai metodi già definiti
    this.kernel=kernel;
    //La directory di upload verrà utilizzata dalla
    //classe Java File quindi il formato dovrà essere
    //quello canonico c:\cartella\sottocartella
    this.uploadDir=kernel.uploadDir; }
  public void run() {
    try {
      //Dopo aver caricato la lista dei file system remoti
      //selezioniamo quale server utilizzeremo per
      //l'upload. Avendo deciso un semplice round robin
      //per schedulare la scelta del file system dovremo
      //mantenere anche l'informazione riguardante
      //l'ultimo server utilizzato
      Vector list=kernel.fslist;
      String url=(String)list.elementAt(
                      kernel.lastpos%list.size());
      //Carichiamo quindi la directory di upload e ci
      //facciamo restituire la lista dei file presenti.
      //Per ognuno di essi richiameremo il metodo copia,
      //utilizzando come parametri il path locale e l'url
      //remoto. Successivamente cancelliamo il file dalla
      //cartella
      File f=new File(uploadDir);
      File lista[]=f.listFiles();
      for ( int i = 0; i < lista.length; i++ ){
        System.out.println( lista[i].getAbsolutePath() );
        copia(lista[i].getAbsolutePath(),url);
        lista[i].delete();}
      if (lista.length>0)
        kernel.lastpos++; }
      catch(Exception e) {
        System.out.println(e.toString());} }
}
```

Grazie a questa classe ogni cinque secondi verrà controllata la cartella che abbiamo scelto come transito per i file e verranno trasferiti i file sui repository remoti. Possiamo dire che la parte centrale del programma è stata sviluppata. Ora dobbiamo occuparci di reperire la lista dei file system da un server.

LATO SERVER: AUTENTICAZIONE

Esiste un server dove vengono conservate tutte le informazioni riguardanti utenti e liste di risorse per ogni utente. Questo serve appunto alla nostra applicazione per poter distribuire ed aggiornare le liste. Il server prima di effettuare qualsiasi operazione deve riconoscere l'utente, quindi dobbiamo avere un database di utenti, definiti nella seguente maniera

```
CREATE TABLE utenti (id INTEGER GENERATED BY DEFAULT
        AS IDENTITY(START WITH 0 , INCREMENT BY 1 ),
    user VARCHAR NOT NULL, pass VARCHAR NOT NULL ,
                                PRIMARY KEY (id ) );
```

Oltre al database degli utenti dovremo anche avere un database con la lista delle risorse per ogni utente. Chiaramente in questo database dovremo riferirci alla lista degli utenti, altrimenti non sapremmo di chi è la risorsa che viene elencata. Dovremo quindi inserire un campo per l'id dell'utente

```
CREATE TABLE risorse ( id INTEGER GENERATED BY
DEFAULT AS IDENTITY(START WITH 0 , INCREMENT BY 1
        ), utente INTEGER, url VARCHAR NOT NULL ,
            PRIMARY KEY (id), FOREIGN KEY(utente)
                            REFERENCES utenti(id) );
```

Ora che abbiamo creato le due tabelle dobbiamo supporre che ci sia un'interfaccia web dove gli utenti possono iscriversi. Quindi pensando di avere già degli utenti inseriti scriviamo la Servlet che in base ad utente e password restituirà la lista di risorse. Nel metodo *doGet()* della nostra Servlet dovremo distinguere due possibili azioni: autenticazione e l'inserimento di una nuova risorsa. Ecco quindi la fase iniziale di autenticazione

```
String action=request.getParameter("action");
if (action.equals("autenticazione")) {
String user=request.getParameter("user");
  String pass=request.getParameter("pass");
  Vector list=caricaLista(user,pass);
  if (list.size()>0) {
    for(int i=0; i<list.size() ; i++) {
      out.print(list.elementAt(i)+";"); } }
    else out.print("0");
}
```

Il metodo *caricaLista()* carica appunto la lista delle risorse associate all'utente che si è collegato. Se non dovesse esistere l'utente o se la password fosse sbagliata il metodo ritornerebbe un vettore vuoto e quindi la risposta del server sarebbe *0*, che il client poi interpreterà come un errore. La connessione al database avviene tramite JDBC, settando nel classpath le librerie relative al database da utilizzare.
In questo caso è stato usato *HypersonicSQL*.

```
public Vector caricaLista(String user,String pass) {
    Connection c;
    Statement SQLStatement;
    ResultSet rs;
    Vector liste=new Vector();
    try {
      Class.forName("org.hsqldb.jdbcDriver" );
```

APPROFONDIMENTO

Come già detto Common VFS utilizza tante librerie presenti sul sito di Jakarta. Per una lista completa di quelle che servono vi rimando all'url http://jakarta.apache.org/commons/sandbox/vfs/download.html

È seguire lo sviluppo di Common VFS e di tanti altri progetti opensource che potete trovare sul sito di Jakarta grazie alle mailing list. Sono disponibili due diverse tipologie di mailing list, utenti e sviluppatori. Per ulteriori informazioni consultate il sito http://jakarta.apache.org/site/mail.html

```
    c = DriverManager.getConnection("jdbc:hsqldb:
                hsql://localhost/fsp", "user", "pass");
  } catch (Exception e) {
      e.printStackTrace();
      return null; }
   try {
      SQLStatement = c.createStatement();
      rs = SQLStatement.executeQuery("SELECT url
       FROM risorse WHERE utente IN (select id from
                utenti where user='"+user+"' and
                        pass='"+pass+"'); ");
      while(rs.next())
      liste.add(rs.getString(1));
      if(rs != null) rs.close();
      if(SQLStatement != null) SQLStatement.close();
      if(c != null) c.close();
   } catch (Exception e) {
     e.printStackTrace();
     return liste; }
  return liste; }
```

Con una semplice connessione al database ci facciamo restituire, se esiste, la lista di risorse associate all'utente e la restituiamo nella risposta.
Quindi l'autenticazione e la trasmissione delle risorse da server a client è stata implementata

INSERIMENTO RISORSA

Ora dobbiamo occuparci di un'altra funzionalità base per il nostro programma, ovvero l'inserimento di una nuova risorsa per un utente. All'inizio avremo soltanto un utente registrato e zero risorse associate. Poi piano piano il client inserirà delle risorse nel database, che il nostro programma popolerà di file nella maniera che abbiamo già discusso. Dobbiamo prevedere un ulteriore switch nella nostra Servlet, passando un diverso parametro come *"action"*.

```
else if (action.equals("inserimento")) {
   String user=request.getParameter("user");
   String pass=request.getParameter("pass");
   String risorsa=request.getParameter("risorsa");
   boolean inserito=inserisciRisorsa(user,pass,risorsa);
   if (inserito) {
      out.print("1");}
   else out.print("0");}
```

rimane da definire il metodo per inserire la risorsa dell'utente all'interno del database. Per farci restituire tutte le risorse associate ad un utente dobbiamo effettuare una connessione simile alla precedente, trovando prima di tutto l'id relativo all'utente che vuole inserire una nuova risorsa e poi effettuare un *INSERT* sulla tabella delle risorse con l'id ottenuto.

```
int id=-1;
SQLStatement = c.createStatement();
rs = SQLStatement.executeQuery("select id from
  utenti where user='"+user+"' and pass='"+pass+"'; ");
while(rs.next())
id=rs.getInt(1);
rs = SQLStatement.executeQuery("insert into
      risorse(UTENTE,URL) values('"+id+"', '"+url+"')"
```

Con questo ultimo metodo della Servlet abbiamo completato la parte server del nostro programma. In teoria servirebbe gestire anche gli utenti, la registrazione, l'eliminazione. Ma per lo scopo di questo articolo ci fermiamo a questo punto.

INTERFACCIA GRAFICA

Dobbiamo implementare l'interfaccia grafica attraverso la quale l'utente può utilizzare il nostro programma. Viste le poche (ma corpose) funzionalità che sono inserite all'interno dell'applicazione viene d'istinto pensare ad una semplice interfaccia grafica. Inizialmente dobbiamo porre l'utente davanti a tre possibili funzioni: la sincronizzazione, l'upload di un file e l'inserimento di una nuova risorsa. Pensiamo quindi di far visualizzare un JFrame, all'interno del quale disporremo tre bottoni con tre diverse immagini.

```
super("FSp v0.0.1");
setSize(250, 150);
setLocation(300,200);
JPanel panel=new JPanel();
panel.setLayout(new GridLayout(1,2));
ImageIcon icon1 = new ImageIcon("Dl.gif");
ImageIcon icon2 = new ImageIcon("Ul.gif");
ImageIcon icon3 = new ImageIcon("Insert.gif");
ButtonActionListener bal1=new
                    ButtonActionListener("Download");
ButtonActionListener bal2=new
                      ButtonActionListener("Upload");
ButtonActionListener bal3=new
                   ButtonActionListener("Inserisci");
JButton uno=new JButton(icon1);
JButton due=new JButton(icon2);
JButton tre=new JButton(icon3);
uno.addActionListener(bal1);
due.addActionListener(bal2);
tre.addActionListener(bal3);
panel.add(uno);
panel.add(due);
panel.add(tre);
getContentPane().add("Center",panel);
show();
```

Come si può vedere, su ogni bottone abbiamo aggiunto un *ButtonActionListener.* Questa è un classe definita da noi che implementa l'interfaccia *Action-*

SUL WEB

Ecco alcuni link riguardanti le tecnologie supportate da Common VFS

http://samba.org/cifs/
http://www.microsoft.com/mind/1196/cifs.asp
http://www.freesoft.org/CIE/RFC/959/index.htm
http://www.webdav.org/
http://en.wikipedia.org/wiki/SSH_file_transfer_protocol

CONTATTA L'AUTORE

L'autore, Federico Paparoni, può essere contattato per suggerimenti o delucidazioni all'indirizzo email federico.paparoni@javastaff.com

Listener e che permette di sapere quando un bottone è stato premuto. In base quindi al bottone che viene cliccato dall'utente dovremo implementare un diverso metodo

```
class ButtonActionListener implements ActionListener {
   String action;
   public ButtonActionListener(String action) {
      this.action=action;}
   public void actionPerformed(ActionEvent e) {
      if (action.equals("Download"))
         k.synch();
      else if (action.equals("Upload"))
         chooseUpload();
      else
         insertRisorsa(); } }
```

Tramite questi tre diversi metodi riusciamo a dare all'utente il completo controllo dell'applicazione. Il metodo relativo al download non farà altro che sincronizzare un'altra volta il contenuto della nostra cartella locale, richiamando un metodo della classe Kernel. Il metodo *chooseUpload()* permette invece all'utente, tramite un'interfaccia grafica, di scegliere quale file selezionare per l'upload. In questo modo automatizziamo il fatto di dover andare a copiare il file nella nostra cartella di transito.

```
public void chooseUpload() {
  try  {
     String filename = File.separator+"tmp";
     JFileChooser fc = new JFileChooser(new
                                         File(filename));
     // Mostra la finestra per la scelta del file
     fc.showOpenDialog(this);
     File selFile = fc.getSelectedFile();
     System.out.println(selFile.getCanonicalPath());
     k.upload(""+selFile.getCanonicalPath());}
  catch(Exception e) {System.out.println(e.toString());}}
```

Infine il metodo *insertRisorsa()* chiede all'utente quale risorsa vogliamo inserire sul server, utilizzando la classe grafica *JOptionPane* .

TESTING

Per testare la nostra applicazione dobbiamo prima di tutto mettere nel *CLASSPATH* tutte le librerie richieste da Common VFS (e sono parecchie!!), poi una volta compilata dobbiamo creare un file di Property inserendo i nostri dati. Infine dobbiamo fare il deploy del *WAR (Web Archive)* della Servlet nella sottocartella *webapps* di Tomcat. Fate partire il programma e il gioco è fatto.

COMUNICAZIONE CLIENT-SERVER

Eccoci quindi arrivati all'ultima parte del programma. Praticamente manca la classe che si occuperà di gestire la comunicazione con il server, prima identificandosi e prendendo la lista di risorse e poi inserendo nuove risorse sul server. Per il momento username e password vengono lasciati in un file di Property, ma chiaramente si potrebbe pensare ad una migliore gestione di questi dati.
Quindi, adesso, all'inizio del programma, verranno recuperate dal file di Property le informazioni necessarie al nostro programma

```
Properties p=new Properties();
p.load(new FileInputStream("fsp.txt"));
String user=p.getProperty("user");
String pass=p.getProperty("pass");
String downloadDir=p.getProperty("downloadDir");
String uploadDir=p.getProperty("uploadDir");
k=new Kernel(user,pass,downloadDir,uploadDir);
```

Per la comunicazione con il server dovremo creare una classe, *Comunication*, che una volta inizializzata con username e password potrà essere interrogata per dialogare con il server. Inizializziamo la classe all'interno del Kernel, poi successivamente la richiamiamo in base all'input fornito dall'utente.
Il primo metodo da definire è quello riguardante la lista di risorse

```
//Qui avviene la connessione alla Servlet, passando come
//argomento l'action autenticazione e i dati relativi all'utente
URL server = new URL("http://127.0.0.1:8080
    /PFSweb/PFSServlet?action=autenticazione&user=
                              "+user+"&pass="+pass+"");
BufferedReader in = new BufferedReader(new
             InputStreamReader(server.openStream()));
String inputLine;
String input="";
Vector el=new Vector();
while ((inputLine = in.readLine()) != null) {
   input=input+inputLine; }
//Dopo aver letto tutta la risposta del server noi sappiamo
//che le risorse sono formattate nel seguente modo:
//risorsa1;risorsa2;risorsa3; Quindi utilizziamo uno
//StringTokenizer per avere tutte le risorse e per
//restituirle al Kernel sotto forma di vettore
StringTokenizer st = new StringTokenizer(input,";");
while (st.hasMoreTokens()) {el.add(st.nextToken());}
in.close();
return el;
```

Nell'altro metodo che conterrà la classe *Comunication* dovremo solo comunicare al server qual è la nuova risorsa da inserire, richiamando semplicemente la Servlet con i giusti parametri. Ecco quindi completato tutto il nostro progetto.

CONCLUSIONI

Ci sono tantissime cose che possono essere fatte per migliorare un software del genere. Prima di tutto l'interfaccia grafica. Poi la gestione dei dati personali, che adesso è relegata ad un file di testo. Il nostro scopo era comunque metervi in grado di utilizzare la potenza di Jakarta Common VFS. Speriamo di esserci riusciti.

Federico Paparoni

Creiamo un OCR con Visual Basic .NET

L'installazione di Office 2003 ha come effetto secondario quello di aggiungere al sistema alcune simpatiche librerie. Una in particolare ci servirà per creare un nostro modulo per il riconoscimento del testo

Quando installiamo una nuova release di Office a quasi tutti capita di non far troppo caso alle applicazioni accessorie in essa contenute. Confesso che anche a me era a prima vista sfuggita una piccola applicazione accessoria che a noi programmatori può dare molte soddisfazioni. Tra gli strumenti presenti nella versione Office 2003 c'è un piccolo programma chiamato *Microsoft Office Document Imaging* che serve a effettuare il riconoscimento caratteri (*OCR* per gli amici) su documenti digitalizzati in formato *TIFF.*

E fin qui, direte voi, buono a sapersi. Ma il bello è che la versione di *Microsoft Office Document Imaging* integrata in office 2003 (non quella di Office XP) rende disponibile nel sistema anche una libreria COM con relative API.

A questo punto lo scaltro programmatore avrà già intuito il gioco: sfruttare le librerie di *Microsoft Office Document Imaging*, che a questo punto potremmo cominciare anche a chiamare confidenzialmente *MODI*, per integrare un *OCR* nelle nostre applicazioni! In particolare vedremo come utilizzare *MODI* nell'ambiente di sviluppo .NET.

IL NOSTRO PROGETTO OCR

La prima cosa da fare è referenziare nel progetto Visual Basic di Visual Studio, come illustrato nei passi, la libreria *"Microsoft Office Document Imaging 11.0 Type Library"* che si dovrebbe trovare nel tab *"COM"* della finestra di dialogo che appare cliccando su *soluzione/references/aggiungi riferimento* (se non fosse presente o non abbiamo installato Office 2003 nel sistema oppure la funzionalità non è stata prescelta in fase di installazione della suite). A questo punto Visual Studio crea per noi una libreria wrapper che espone in .Net metodi e proprietà dell'oggetto COM sottostante. In un successivo passaggio dobbiamo poi inserire nella *Casella* degli strumenti il riferimento al controllo ActiveX che ci servirà per ottenere un viewer nel quale l'utente può compiere visivamente le operazioni di *OCR* sul documento. L'operazione è simile alla precedente con la differenza che dobbiamo partire con un click con il tasto destro sulla casella degli strumenti e poi selezionare la voce *"aggiungi /rimuovi elementi"* nel menu contestuale. Anche qui scegliamo *Microsoft Office Document Imaging Viewer Control* nel tab *"COM"* dei controlli come possiamo vedere in **Figura 1** e vedremo comparire nella casella degli strumenti, insieme agli altri controlli *Windows Forms* anche l'icona del controllo *MODI Viewer*).

Naturalmente l'utilizzo del controllo ActiveX è necessario solo se, come nel nostro caso, vogliamo disporre di un'interfaccia utente per la gestione di *MODI,* si potrebbero avere infatti dei progetti che utilizzano *MODI* per processare automaticamente dei documenti.

Fig. 1: **Inserimento del controllo ActiveX**

IL MODELLO AD OGGETTO ESPOSTO DA MODI

Il modello ad oggetti di MODI consiste nei seguenti oggetti principali:

- **DOCUMENT rappresenta una *ordered collection* di pagine (*images*).**
- **IMAGE rappresenta la singola pagine di un documento.**
- **L'oggetto LAYOUT espone i risultati del riconoscimento caratteri su una pagina.**
- **MIDOCSEARCH espone le funzionalità di ricerca sul documento.**
- **Il controllo VIEWER (l'oggetto *MiDocView*) è un controllo ActiveX che mostra le pagine di un documento.**

Fig. 2: *Il controllo ActiveX nella casella degli strumenti*

L'ACQUISIZIONE DA SCANNER

I driver di acquisizione da scanner e da altre periferiche in ambiente Windows seguono soprattutto due standard: TWAIN e WIA (Windows Image Acquisition Driver).

***TWAIN* è un API per l'acquisizione di immagini per i sistemi operativi Microsoft Windows e Apple Macintosh. La prima specifica per questo standard fu definita nel 1992, ed attualmente viene seguita la versione 1.9 rilasciata a Gennaio 2000. TWAIN è tipicamente utilizzato come un'interfaccia tra il software applicativo ed uno scanner o fotocamera digitale. La parola TWAIN è ripresa dalla "*Ballata dell'Est e dell'Ovest*" di Kipling - "*...and never the twain shall meet...*", che riflette la difficoltà, di connettere scanner e personal computer. La parola TWAIN è stata utilizzata per rappresentare lo standard con notazione maiuscola. Questo erroneamente ha portato a ritenerla un acronimo che alcuni hanno interpretato ironicamente come "*Technology Without An Interesting Name*". Il sito ufficiale di TWAIN è www.twain.org.**

***WIA* è invece un API proprietaria di Microsoft che persegue più o meno gli stessi scopi presentandosi come interfaccia di programmazione la *"Still Image (STI) architecture"* introdotta con Windows Me, Windows XP, e piattaforme successive. Informazioni su WIA sono reperibili sul sito : http://msdn.microsoft.com/library/default.asp?url=/library/en-us/still/hh/still/WIA_intro_1b17ef5f-807a-4077-9cc3-e8e33b178bf1.xml.asp. Nessuna delle due tecnologie dispone di classi dedicate nel .NET framework, tuttavia sono stati sviluppati due progetti: www.codeproject.com/dotnet/twaindotnet.asp per TWAIN e www.codeproject.com/dotnet/wiascriptingdotnet.asp per WIA che dimostrano la possibilità di utilizzarle anche dal framework.**

CHIAMARE MODI...

Una volta che abbiamo creato i riferimenti necessari nel progetto vediamo cosa bisogna fare per utilizzare praticamente la libreria *MODI*. Innanzitutto creiamo una, per quanto minimale, interfaccia utente composta da:

- una Form
- un menu principale con i comandi di base
- una barra di status dove visualizzare i messaggi del programma
- e naturalmente ... un controllo *MODI Viewer*.

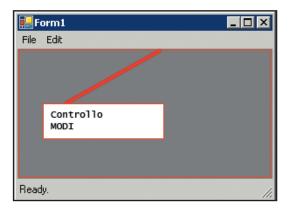



Fig. 3: *L'interfaccia utente con evidenziato il controllo ActiveX disegnato sulla Form*

Adesso che abbiamo tutti gli strumenti vediamo come interagire da codice con le funzionalità di *OCR*. Quello che dovrà fare il nostro programma di prova sarà:

1. consentire all'utente di scegliere un file su cui applicare l'*OCR*;
2. gestire il processo di *OCR* segnalandone lo stato;
3. copiare o salvare l'output prodotto.

Omettiamo per brevità il primo passaggio rimandando al codice allegato nel CD l'analisi dei dettagli implementativi. Passiamo direttamente al punto in cui disponiamo di un file *TIFF* scelto dall'utente (o meglio del percorso in cui questo file è memorizzato) , alla selezione dovremmo costruire un metodo simile a:

```
Private _MODIDocument As MODI.Document = Nothing
   Private Sub SetImage(ByVal filename As String)
      ' prepara l'immagine..
      Try
         _MODIDocument = New MODI.Document
         _MODIDocument.Create(filename)
         AxMiDocView1.Document = _MODIDocument
         AxMiDocView1.Refresh()
      Catch ex As System.Runtime
                  .InteropServices.COMException
      MessageBox.Show(ex.Message)
   End Try
End Sub
```

Si valorizza cioè un oggetto del tipo *MODI.Document* dichiarato a livello di classe attivandone il metodo *Create* passandogli il riferimento al percorso del file *TIFF.* Successivamente assegniamo il *Document* appena creato alla relativa proprietà del controllo ActiveX chiamando inoltre il metodo *Refresh* di quest'ultimo.
A questo punto il documento sarà *_MODIParameters* visibile all'utente nel controllo ed in fase di esecuzione apparirà l'anteprima che possiamo vedere in **Figura 4**.
A questo punto scriviamo il metodo che fa partire il processo di riconoscimento del testo:

Utilizza questo spazio per le tue annotazioni

IL FORMATO TIFF

Il formato TIFF *(Tag Image File Format)* è il formato in cui vengono archiviati di solito le immagini ottenute dagli scanner o ricevute via Fax. È un formato non distruttivo che permette la manipolazione dei singoli canali colore e dà luogo a files di grosse dimensioni.

FORMATI DI IMMAGINE GESTITI DA MODI

Stando alla documentazione sembrerebbe che gli unici formati su cui MODI sia in grado di operare il riconoscimento caratteri siano *TIFF* e *MDI* (un formato proprietario), in realtà nelle nostre prove siamo riusciti a caricare nel *MODI Viewer Control* anche immagini di numerosi altri formati grafici tra cui *JPG, GIF* e *BMP.*

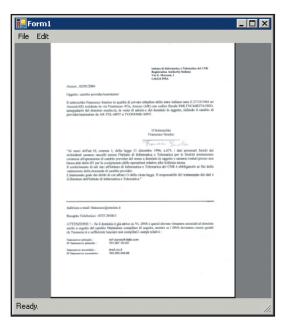
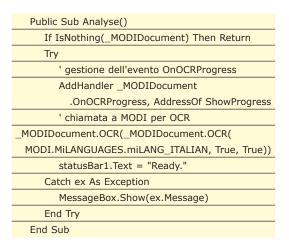



Fig. 4: ***L'anteprima del documento caricata nel Viewer Control***

```
Public Sub Analyse()
    If IsNothing(_MODIDocument) Then Return
    Try
        ' gestione dell'evento OnOCRProgress
        AddHandler _MODIDocument
            .OnOCRProgress, AddressOf ShowProgress
        ' chiamata a MODI per OCR
_MODIDocument.OCR(_MODIDocument.OCR(
  MODI.MiLANGUAGES.miLANG_ITALIAN, True, True))
        statusBar1.Text = "Ready."
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub
```

Il metodo è semplicissimo: si aggiunge un gestore dell'evento *OnOCRProgress* del *MODIDocument* (che non è altro che l'oggetto che abbiamo visto valorizzare da *SetImage*), si chiama il metodo *OCR* del *MODIDocument* stesso e si segnala la fine dell'operazione nella *StatusBar.*
I parametri di configurazione dell'operazione di *OCR* assegnati al metodo sono tre:

1. **LangId** *(opzionale)* rappresenta la lingua da utilizzare per l'operazione di *OCR*, come predefinita è *miLANG_SYSDEFAULT* ovvero quella di sistema. Le lingue riconosciute dall'engine di *MODI* sono quelle corrispondenti all'enumerazione *MiLANGUAGES:*

   - *miLANG_CHINESE_SIMPLIFIED (2052, &H804)*
   - *miLANG_CHINESE_TRADITIONAL (1028, &H404)*
   - *miLANG_CZECH (5)*
   - *miLANG_DANISH (6)*
   - *miLANG_DUTCH (19, &H13)*
   - *miLANG_ENGLISH (9)*
   - *miLANG_FINNISH (11)*
   - *miLANG_FRENCH (12)*
   - *miLANG_GERMAN (7)*
   - *miLANG_GREEK (8)*
   - *miLANG_HUNGARIAN (14)*
   - *miLANG_ITALIAN (16, &H10)*
   - *miLANG_JAPANESE (17, &H11)*
   - *miLANG_KOREAN (18, &H12)*
   - *miLANG_NORWEGIAN (20, &H14)*
   - *miLANG_POLISH (21, &H15)*
   - *miLANG_PORTUGUESE (22, &H16)*
   - *miLANG_RUSSIAN (25, &H19)*
   - *miLANG_SPANISH (10)*
   - *miLANG_SWEDISH (29, &H1D)*
   - *miLANG_SYSDEFAULT (2048, &H800)*
   - *miLANG_TURKISH (31, &H1F)*

2. **OCROrientImage** - *(opzionale Boolean)* Modalità di correzione automatica della rotazione del documento. Come valore predefinito assume True.

3. **OCRStraightenImage** - *(opzionale Boolean)* Correzione delle distorsioni dell'immagine. Come valore predefinito assume *True.*

Ovviamente nel nostro progetto dovremo chiamare il metodo *Analyse* da un gestore di eventi quale, ad esempio il metodo che gestisce il click su una voce di menu:

```
Private Sub miAnalyse_Click(ByVal sender As Object,
                ByVal e As System.EventArgs) Handles
                                        miAnalyse.Click
    Analyse()
End Sub
```

COPIARE E SALVARE

Una volta compiuto il processo di *OCR* il testo riconosciuto resta associato al file originario tanto che l'utente può selezionare tutto o parte del testo

Fig. 5: ***La selezione del testo riconosciuto nell'anteprima del documento***

nella finestra di anteprima come vediamo in **Figura 5**.
Naturalmente questa associazione è gestibile da codice attraverso la proprietà *TextSelection* del controllo *MODI Viewer* che restituisce un oggetto del tipo *IMiSelectableItem* che espone i seguenti metodi/proprietà a noi utili:

- Metodo **CopyToClipboard** che naturalmente copia il testo selezionato negli appunti di sistema.
- Proprietà **Text** che rappresenta la stringa di testo riconosciuta.
- Proprietà **Words** che rappresenta un array di oggetti *MODI.Word* , in pratica corrispondenti alle singole parole del testo riconosciuto.

A questo punto a noi non resta che dare la possibilità all'utente di salvare in un file o negli appunti di sistema il testo riconosciuto dall'engine *OCR*.
Lo facciamo con due semplici metodi associati anch'essi a voci di menu. Per salvare negli appunti:

```
Private Sub miCopy_Click(ByVal sender As Object,
                         ByVal e As System.EventArgs)
                         Handles miCopy.Click
    ' copia nella clipboard..
    If IsNothing(AxMiDocView1.TextSelection) Then
        AxMiDocView1.SelectAll(0)
    End If
    AxMiDocView1.TextSelection.CopyToClipboard()
End Sub
```

Dove se l'utente non ha selezionato nessuna porzione di testo si fa partire automaticamente una selezione di tutta la prima pagina con il metodo *SelectAll* del *viewer control* passandogli come parametro il numero di pagina (in questo caso *0* come primo elemento dell'array delle pagine).

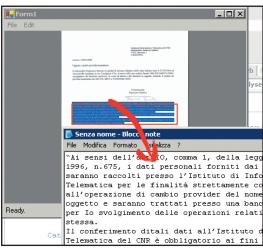



Fig. 6: ***Il risultato del nostro OCR copiato in notepad***

Dopodiché si richiama semplicemente il metodo *CopyToClipboard* sulla selezione.
Altrettanto semplice il codice per salvare su file:

```
Private Sub miSave_Click(ByVal sender As Object,
                         ByVal e As System.EventArgs)
                         Handles miSave.Click
    ' salva ..
    If Not IsNothing(_MODIDocument) Then
        If IsNothing(AxMiDocView1.TextSelection) Then
            AxMiDocView1.SelectAll(0)
        End If
        Dim TextSelection As String =
                    AxMiDocView1.TextSelection.Text()
    ' ometto il codice per creare la finestra di dialogo ..
        If saveFileDlg.ShowDialog(Me) =
                    DialogResult.OK Then
            Dim writer As IO.StreamWriter
            writer = IO.File.CreateText(
                    saveFileDlg.FileName)
            writer.Write(TextSelection)
            writer.Close()
        End If
    End If
End Sub
```

Dove si acquisisce la stringa riconosciuta tramite la proprietà *text* della selezione e la si scrive in un file con i metodi standard di *IO* di .Net.

CONCLUSIONI

La libreria *MODI* si presta naturalmente a molte altre (e più sofisticate) applicazioni pratiche: pensate ad esempio ad un sistema di gestione documentale che integri insieme all'archiviazione ottica di un documento anche il riconoscimento del testo che magari può alimentare un database per integrare un motore di ricerca per ritrovare i documenti originari.
Certo in uno scenario di questo tipo *MODI* sconta una grossa limitazione pratica quella di non gestire l'acquisizione del documento da scanner, in realtà l'applicazione vera e propria *Microsoft Office Document Imaging* di Office 2003 ha questa possibilità, purtroppo però (non si sa bene per quale ragione) queste funzionalità non sono state esposte nell'oggetto COM e quindi restano inaccessibili.
Ci sono tuttavia delle interessanti librerie Open Source, scritte in C# e quindi utilizzabili anche da Visual Basic .NET, che presentano delle interfacce agli standard *TWAIN* o *WIA* (vedi box *"L'acquisizione da scanner"*) per l'acquisizione di documenti da scanner, in una prossima occasione ci torneremo sopra!

Francesco Smelzo

Un help completo sulla tecnologia MODI è reperibile all'indirizzo:
http://msdn.microsoft.com/library/en-us/Mspauto/html/dihowUsingMODIObjectModel_HV01049396.asp

L'AUTORE

Francesco Smelzo è specializzato nello sviluppo in ambiente Windows con particolare riferimento ad applicazioni in ambiente .NET sia web-oriented che desktop.
Il suo sito web è www.smelzo.it.

Java PircBot la chat è fatta!

Costruiamo un robot da utilizzare nella gestione di un canale IRC Vedremo come amministrare gli utenti, sviluppare semplici trigger e mostreremo come creare un piccolo dcc server

IRC *(Internet Relay Chat)* è uno dei metodi più diffusi fra i chatters su Internet. Sviluppato nel 1988 è un sistema di chat che si basa su un server centrale e su diversi client che si connettono a questo server seguendo l'RFC 1459. All'interno di questi server esistono diversi canali dove poter entrare e discutere. A loro volta i diversi server entrano in comunicazione fra loro formando una specie di circuito condiviso dagli stessi utenti connessi però a server diversi. Ogni utente sceglie un nickname con il quale viene identificato univocamente su tutti i server. Inoltre c'è la possibilità di gestire e moderare il canale in diverse maniere. Esistono diversi tipi di moderazione: *OP* e *VOICE*. Gli *OP* di un canale sono i proprietari o comunque quelli che gestiscono il canale, cambiano il *TOPIC* (l'argomento), settano limiti e decidono se una persona può rimanere nel canale o no. Invece i *VOICE* sono le uniche persone che possono parlare (oltre agli *OP*) quando il canale è settato in un modalità "moderata". Inoltre esistono diverse versioni di server irc *(ircd)*, tutte aderenti all'RFC 1459 ma con diverse funzionalità. Frequentando i diversi server che sono disponibili in rete si possono incontrare canali che trattano i più svariati argomenti.

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
1:40:50 2005
1114067782904 :irc.foonet.com 004 DeLiRiUm irc.foonet.com Unreal3.2.3 iowghraAsO
RTVSxNCWqBzvdHtGp lvhopsmntikrRcaqOALQbSeIKVfMCuzNTGj
1114067782904 *** Logged onto server.
1114067782914 >>>JOIN #javastaff
1114067782914 :irc.foonet.com 005 DeLiRiUm SAFELIST HCN MAXCHANNELS=10 CHANLIMIT
=#:10 MAXLIST=b:60,e:60,I:60 NICKLEN=30 CHANNELLEN=32 TOPICLEN=307 KICKLEN=307 A
WAYLEN=307 MAXTARGETS=20 WALLCHOPS WATCH=128 :are supported by this server
1114067782914 :irc.foonet.com 005 DeLiRiUm SILENCE=15 MODES=12 CHANTYPES=# PREFI
X=(qaohv)~&@%+ CHANMODES=beI,kfL,lj,psmntirRcOAQKVGCuzNSMTG NETWORK=ROXnet CASEM
APPING=ascii EXTBAN=~,cqnr ELIST=MNUCT STATUSMSG=~&@%+ EXCEPTS INVEX CMDS=KNOCK,
MAP,DCCALLOW,USERIP :are supported by this server
1114067782924 :irc.foonet.com 251 DeLiRiUm :There are 1 users and 1 invisible on
 1 servers
1114067782924 :irc.foonet.com 253 DeLiRiUm 1 :unknown connection(s)
1114067782924 :irc.foonet.com 254 DeLiRiUm 1 :channels formed
1114067782924 :irc.foonet.com 255 DeLiRiUm :I have 2 clients and 0 servers
1114067782924 :irc.foonet.com 265 DeLiRiUm :Current Local Users: 2  Max: 3
1114067782924 :irc.foonet.com 266 DeLiRiUm :Current Global Users: 2  Max: 2
1114067782924 :irc.foonet.com 422 DeLiRiUm :MOTD File is missing
1114067782924 :DeLiRiUm MODE DeLiRiUm :+iwx
1114067782924 :DeLiRiUm!PircBot@6DA1959F.10BF5A3F.D41AEED.IP JOIN :#javastaff
1114067782934 :irc.foonet.com 353 DeLiRiUm = #javastaff :DeLiRiUm @d0c
1114067782934 :irc.foonet.com 366 DeLiRiUm #javastaff :End of /NAMES list.
```

***Fig. 1:** L'output che il nostro bot produce quando si collega ad un server IRC*

Senza perderci nel mare di IRC, delle sue leggende e dei suoi tanti canali passiamo direttamente all'aspetto che analizzeremo in questo articolo. Tramite un framework Java realizzeremo un robot, che è un vero e proprio client del server IRC, vedremo come poter manovrare questo robot dal punto di vista della programmazione e poi lo doteremo di diverse funzionalità interessanti.

PIRCBOT: UN FRAMEWORK PER BOT

PircBot è un framework opensource per sviluppare in Java dei Bot IRC. Grazie a questo framework possiamo in maniera molto semplice "costruire" un Bot irc, ovvero un software che "gestirà" un canale quando noi non siamo connessi. Nel package *org.jibble.pircbot* troviamo la definizione delle classi base del framework, le classi che andremo ad utilizzare. La classe base da cui partire è *PircBot*, estendendo la quale otterremo dei comportamenti specializzati. Vediamo prima di tutto come far partire il nostro primo Bot e farlo collegare al nostro server IRC, in un nostro canale. Il codice da utilizzare è il seguente

```
import org.jibble.pircbot.*;
public class HelloWorldBot extends PircBot {
    public HelloWorldBot(){
        this.setName("HelloWorldBot");}
}
```

Così richiamando il metodo *setName()* abbiamo deciso quale sarà il nickname che il nostro Bot utilizzerà sul server IRC. Il prossimo passo sarà collegare questo Bot ad un server, farlo entrare in un canale, e chiudere la connessione.

```
import org.jibble.pircbot.*;
```

```
public class BotExample {
  public static void main(String a[]){
    HelloWorldBot hwb=new HelloWorldBot();
    hwb.connect("irc.azzurra.org");
    hwb.joinChannel("#javastaff");
    hwb.sendMessage("#javastaff","Ciao a tutti");
    hwb.disconnect(); }
}
```

Con il metodo *connect()* diciamo al Bot a quale server si deve connettere. Una volta che la connessione è avvenuta facciamo entrare il bot nel canale *#javastaff*. Il simbolo # è un prefisso che identifica i canali su IRC (ad esempio *#java, #italia, #roma*). Appena il bot entra nel canale mandiamo un messaggio pubblico su *#javastaff* con il metodo *sendMessage()*. Infine ci disconnettiamo dal server e terminiamo l'esecuzione. Per testare questo primo esempio dobbiamo inserire nel nostro classpath la libreria *pircbot.jar* e chiaramente avere un collegamento ad internet. I server dove poter testare questa applicazione sono tantissimi. Un test per il controllo del buon esito dell'applicazione è molto semplice.
È sufficiente connettersi con un client come mirc - *http://www.mirc.com* - o Xchat - *http://www.xchat.org/* - e vedere cosa combina il nostro Bot.

IL NOSTRO BOT

La cosa più interessante in un Bot è la possibilità di interagire con le persone che sono presenti nel canale. Per fare ciò dobbiamo poter semplicemente controllare i messaggi del canale dove il nostro Bot si trova e parsarli in maniera corretta. In questo caso ci viene in aiuto *PircBot* con la sua libreria. Il formato con cui i messaggi passano sul canale è aderente alla RFC 1459, e i messaggi assomigliano a qualcosa del genere:

1113949468186:d0c!d0c@6DA1959F.10BF5A3F.D41AEED.IP PRIVMSG #javastaff :ciao DeLiRiUm
1113949480624:d0c!d0c@6DA1959F.10BF5A3F.D41AEED.IP PRIVMSG #javastaff :che fai?

Ovviamente un messaggio di questo genere non è adatto ad essere comprensibile. E tipicamente andrebbe "parsato" e "scomposto" per ottenere solo le informazioni che ci servono. Non è necessario riprogrammare il parser usando PircBot, di fatto esiste già il metodo *OnMessage()* nella classe base, che implementa un parser a tutti gli effetti. Attraverso il metodo *OnMessage()* è facile usare dei trigger per riprogrammare a nostro piacimento i comportamenti del Bot. Iniziamo quindi a ridefinire il metodo *onMessage()* per avere una reazione da parte del nostro Bot.

```
public void onMessage(String channel, String sender,
      String login, String hostname, String message) {
  if (message.equals("Africa"))
     sendMessage(channel, "Celestino");
}
```

In questo modo il nostro Bot quando intercetta la parola *"Africa"* all'interno di un messaggio, scriverà come reazione nel canale la parola *"Celestino"*, usando la primitiva *sendMessage()* definita nella classe *PircBot*.

LA GESTIONE DEI CANALI

Come abbiamo visto precedentemente, con il metodo *joinChannel()* possiamo far entrare il nostro Bot nel canale che viene passato come argomento. Immaginiamo ora di volerlo far entrare automaticamente in una serie di canali, definiti magari in un file di testo. Per fare ciò non dobbiamo far altro che scrivere un metodo che, chiamato dopo la connessione al server, legge da un file i nomi dei canali e richiede, uno per uno, di entrare nei canali.

```
public void channelJoin() {
  canali=new Vector();
  try{
    BufferedReader br=new BufferedReader(new
                          FileReader("canali.txt"));
    String linea=br.readLine();
    while(linea!=null) {
      bot.joinChannel("#"+linea);
```

GOOGLE WEB API

Per poter utilizzare le Google Web Api dobbiamo registrarci all'url www.google.com/apis/. Successivamente ci sarà inviata per email la chiave da utilizzare nel nostro programma. Oltre al webservice di Google potrebbero esserne utilizzati tanti altri per includere funzionalità nel nostro Bot. Per questo vi rimando ad uno dei siti dove c'è una raccolta ben fornita di WebService. www.xmethods.net/

GLI ALTRI BOT IRC

Chi ha frequentato per molto tempo IRC avrà sicuramente incontrato dei canali con dei Bot. Nella maggior parte dei casi non si tratta di un Bot sviluppato con PircBot, ma di un suo famoso predecessore, l'Eggdrop. Questo è il primo esempio di Bot su dei server IRC, sviluppato in C e con la possibilità di aggiungere delle estensioni in TCL. Per poter utilizzare un Eggdrop dobbiamo prima di tutto scaricare i sorgenti, compilarli e poi modificare il file di configurazione base. Esistono poi tantissime TCL da aggiungere all'Eggdrop, la maggior parte create per difendere i canali o per il semplice divertimento degli utenti. Un altro Bot molto famoso è l'Iroffer, un Bot che agisce come un vero e proprio file server e che permette di condividere file all'interno dei canali IRC. Ecco un paio di link riguardanti questi due famosi antenati di PircBot
http://www.eggheads.org/
http://www.egghelp.org/
http://iroffer.org/
http://iroffer-lamm .sourceforge.net/

```
        linea=br.readLine();
        canali.add(linea); }
    br.close();}
  catch(Exception e) {}
}
```

Così tutti i canali che avranno l'onore di avere come ospite la nostra creatura saranno facilmente gestibili tramite questo file di testo.

RICONOSCERE GLI UTENTI

Uno dei principali motivi per cui si crea un Bot su dei canali IRC è per gestire il canale, anche quando non sono presenti i veri moderatori del canale. Implementeremo un vero e proprio sistema di identificazione che permetterà soltanto a determinati utenti di utilizzare il nostro Bot e le sue caratteristiche che poi andremo piano piano ad implementare. Avremo bisogno di una classe abbastanza semplice, dove implementeremo i metodi get e set per ogni variabile. Inoltre implementeremo l'interfaccia *Serializable*, così potremo serializzare un vettore di utenti e ricaricarlo tranquillamente all'avvio.

```
import java.util.*;
import java.io.*;
public class User implements Serializable{
  String username,password,email,dir;
  int level;
public User(String username,String password,String
                email,int level,String dir) {
    this.username=username;
    this.password=password;
    this.email=email;
    this.level=level;
    this.dir=dir; }
  public String getUsername() {
    return username;}
  public void setUsername(String username) {
    this.username=username; }
  ...
}
```

Una volta definito come sarà rappresentato l'utente dobbiamo decidere un metodo per far si che gli utenti possano registrarsi. Per fare ciò l'utente dovrà scrivere in query (messaggio privato) al Bot una frase che segue i seguente pattern

```
!REGISTER password email
```

Ridefiniamo il metodo *onPrivateMessage()*, creando prima di tutto uno *StringTokenizer* basato sullo spazio, per ottenere così tutte le singole parole presenti nel messaggio che ci viene passato.

```
public void onPrivateMessage(String sender, String
            login, String hostname, String message) {
  StringTokenizer st=new StringTokenizer(message," ");
  String action=st.nextToken();
  //REGISTRAZIONE UTENTE
  if(action.equals("!REGISTER")) {
    registerAction(st,sender); }
}
```

COME INIZIARE

> CREAZIONE DEL BOT

```
public class Delirium extends PircBot
...
...
Delirium bot = new Delirium();
bot.setVerbose(true);
bot.setName("DeLiRiUm");
```

1 Dopo aver incluso la libreria pircbot.jar nel nostro classpath creiamo la classe che estende PircBot e nel main del nostro programma inizializziamo il Bot.

> CONNESSIONE AL SERVER

```
bot.connect("server.irc");
canali=new Vector();
try{
  BufferedReader br=new BufferedReader(
          new FileReader("canali.txt"));
  String linea=br.readLine();
  while(linea!=null) {
    bot.joinChannel("#"+linea);
    linea=br.readLine();
    canali.add(linea); }
  br.close();}
catch(Exception e) {
}
```

2 Colleghiamo il Bot al server IRC passato come parametro al metodo connect(). Effettuata la connessione entriamo in tutti i canali che sono presenti in un nostro di testo.

> LA PRIMA PAROLA

```
public void onMessage(
 String channel, String sender, String login,
           String hostname, String message)
  {
    if (message.equals("Africa"))
      sendMessage(channel, "Celestino");
  }
```

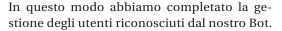
3 Facciamo pronunciare la prima parola alla nostra creatura. Implementando il metodo onMessage() decidiamo un trigger, in risposta al quale il Bot ci risponde nel canale.

In questo modo registriamo l'utente aggiungendolo al vettore degli utenti. Ora rimangono da scrivere i due metodi che all'inizio e alla fine dell'esecuzione del nostro Bot caricherà e salverà il vettore di utenti registrati, serializzando su file. All'avvio quindi eseguiremo il seguente metodo

```
public void caricaUtenti() {
  try {
    FileInputStream in = new FileInputStream(
                                    "user.ser");
    ObjectInputStream s = new
                          ObjectInputStream(in);
    utenti =(Vector) s.readObject();
    s.close(); }
  catch(Exception e) {
    System.out.println(e.toString()); }
}
```

Così abbiamo caricato nel vettore tutti gli utenti registrati. Ora definiremo il metodo che invece sarà richiamato prima della disconnessione del nostro Bot dal server IRC, per salvare sul file il vettore di utenti aggiornati. Anche in questo caso utilizziamo uno stream che ci permette di manipolare direttamente oggetti come *ObjectOutputStream* (e in precedenza *ObjectInputStream*).

```
public void salvaUtenti() {
  try {
    FileOutputStream out = new
                    FileOutputStream("user.ser");
    ObjectOutputStream s = new
                        ObjectOutputStream(out);
    s.writeObject(utenti);
    s.close(); }
  catch(Exception e) {
    System.out.println(e.toString()); }
}
```

In questo modo abbiamo completato la gestione degli utenti riconosciuti dal nostro Bot.

INVIO E RICEZIONE FILE

Utilizzando IRC è possibile effettuare uno scambio diretto di file che viene chiamato *DCC (Direct Client to Client)*, utilizzato anche per poter parlare. Chiaramente anche il nostro Bot può utilizzare queste funzionalità, essendo lui stesso un client che si collega al server IRC. Prima occupiamoci di ricevere i file. Esiste un metodo definito all'interno della classe *PircBot* che ci permette di avere la notifica del file in arrivo. Questo metodo, *onIncomingFileTransfer()*, dovrà essere implementato nel nostro Bot, per poter accettare i file in arrivo.

```
public void onIncomingFileTransfer(DccFileTransfer
                                  transfer) {
  File file = transfer.getFile();
  transfer.receive(file, true);
 }
```

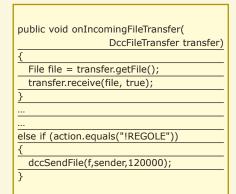
Il file che abbiamo accettato di ricevere verrà direttamente copiato nella cartella dove è in esecuzione il nostro Bot. In questo modo potremmo anche creare un repository per tutti

> CARICARE GLI UTENTI

```
public void caricaUtenti() {
  try {
    FileInputStream in = new
                 FileInputStream("user.ser");
    ObjectInputStream s = new
                       ObjectInputStream(in);
    utenti =(Vector) s.readObject();
    s.close();
  }
  catch(Exception e) {
    System.out.println(e.toString());
  }
}
```

4 Carichiamo una lista di utenti che si sono precedentemente registrati. Andiamo a leggere il vettore degli utenti direttamente da un file, dove li abbiamo salvati nella precedente sessione.

> GESTIONE DEI FILE

```
public void onIncomingFileTransfer(
                  DccFileTransfer transfer)
{
  File file = transfer.getFile();
  transfer.receive(file, true);
}
...
...
else if (action.equals("!REGOLE"))
{
  dccSendFile(f,sender,120000);
}
```

5 Per ricevere un file dobbiamo implementare *onIncomingFileTransfer()* e accettare il trasferimento. Per l'invio utilizziamo *dccSendFile()*, indicando l'utente, il file e il timeout.

> RICERCHE CON GOOGLE

```
WebSearch ws=new WebSearch(0);
Vector result=ws.search(toSearch);
sendMessage(channel,sender+" ecco
            i risultati della tua ricerca");
for (int i=0;i<result.size();i++)
{
  String temp=(String)result.elementAt(i);
  sendMessage(channel,temp);
}
```

6 Creiamo una class WebSearch utilizzando le Google Web Api. Passiamo come parametro le parole inviate dall'utente nel canale. Comunichiamo i risultati tramite la primitiva *sendMessage()*.

gli utenti in un canale. Ad esempio in un canale che parla di esami universitari si potrebbero salvare le dispense utili per poter passare l'esame. In generale la tecnica per inviare file è simile a quella della ricezione solo che dobbiamo indicare una persona alla quale inviare il file. In questo caso permetteremo al nostro Bot di essere attivato dal trigger *!REGOLE* per poter inviare un file con le regole del canale a chi le ha richieste. Inseriamo quindi il trigger nel metodo *onMessage()* che già abbiamo implementato e poi inviamo direttamente un file di regole caricato all'avvio del programma

```
else if (action.equals("!REGOLE")) {
            dccSendFile(f,sender,120000);
}
```

Da notare che noi dobbiamo semplicemente richiamare il metodo *dccSendFile()* passando come parametro il file, chi ha attivato il trigger *!REGOLE* e un timeout per la richiesta.

PircBot è scaricabile dal sito http://www.jibble.org/pircbot.php **dove potete trovare della documentazione e molti link a progetti che hanno utilizzato questo framework per sviluppare un Bot.**

ALTRE FUNZIONALITÀ DI BASE

Oltre a quello che abbiamo già implementato dobbiamo permettere al nostro Bot di svolgere le normali funzioni di utente IRC. Nel nostro caso specifico è interessante vedere come possiamo comandarlo per rendere altri utenti *OP* del canale o levare l'*OP* del canale a qualcuno. Nei canali IRC che non abbiano una gestione tramite Bot o tramite particolari funzionalità di registrazione del canale sull'IRCD ci sono elevate probabilità di perdere il canale a causa di scorribande di persone che cercano di rubarlo. Per gestire queste funzionalità base ci sono delle semplici metodi, già definiti in *PircBot*, che devono essere semplicemente richiamati. Quindi come già abbiamo fatto in precedenza inseriamo dei trigger nel metodo *onMessage()*

```
if (action.equals("!OP")) {
   opAction(st,sender,channel); }
else if(action.equals("!DEOP")) {
   deopAction(st,sender,channel); }
```

Poi definiremo i due metodi che prima di eseguire l'ordine eseguiranno un controllo sull'utente. Infatti dobbiamo chiaramente controllare che l'utente sia autorizzato ad eseguire determinate azioni. Questo controllo è possibile facendo identificare l'utente in query (messaggio privato) col Bot. In questo messaggio l'utente comunicherà al Bot la sua password e il Bot vedendo se l'utente è registrato lo inserirà tra gli utenti identificati. Ecco quindi la definizione di uno dei due metodi richiamati prima

```
public void opAction(StringTokenizer st,String
                                        sender,String channel) {
   String who=st.nextToken();
   if (utentiIdentificati.contains(sender))
      op(channel,who);
}
```

Richiamando il metodo *op()* (e il metodo *deop()* nell'altro caso) comunichiamo al server di rendere moderatore l'utente indicato. Chiaramente tutti gli utenti possono registrarsi e identificarsi, quindi potrebbero rubarci il canale. Questa parte può essere implementata in diversi modi, quindi la lascio volutamente non protetta. Infatti potremmo pensare ad una registrazione lato web oppure ad un solo utente (il primo) che può registrare pian piano altri utenti. Questo dipende dalle esigenze del canale che vogliamo gestire e quindi vá oltre allo scopo dell'articolo. Ora che abbiamo implementato delle funzionalità base per il nostro Bot incominciamo a divertirci, portando nel nostro canale dei servizi di ricerca che potrebbero risultare interessanti per gli utenti nel canale.

GOOGLE È TUO AMICO

Abbiamo visto come sia semplice comandare il nostro Bot tramite dei trigger, passandogli anche dei parametri. Ora cerchiamo di sviluppare un'estensione che si interfacci al motore di ricerca Google. L'idea dalla quale vogliamo partire è quella di poter fare le ricerche su Google anche quando sei in chat con altre persone. Per potersi interfacciare con questo motore di ricerca esistono già da tempo le Google Web Api, ovvero delle API che ci permettono di effettuare ricerche su Google grazie ad un WebService che viene esposto. Una volta registrati sul sito *http://www.google.com/apis/* ci viene inviata via email una chiave da utilizzare nel nostro programma. Questa serve per limitare l'accesso al WebService, offrendo massimo 1000 ricerche al giorno. Quindi prima di interfacciare Google con il nostro Bot costruiamoci una classe per effettuare la vera e propria ricerca. Prima di tutto dobbiamo importare i seguenti package

```
import com.google.soap.search.GoogleSearch;
import com.google.soap.search.GoogleSearchResult;
import com.google.soap.search
```

```
.GoogleSearchResultElement;
import com.google.soap.search.GoogleSearchFault;
```

Ora dobbiamo semplicemente istanziare la classe base per le ricerche ovvero *GoogleSearch*

```
GoogleSearch search = new GoogleSearch();
search.setProxyHost("192.168.1.1");
search.setProxyPort(3128);
search.setKey(
        "AhB7sA5QFHJ9ZasdH8ALLLOPOGBeTrQFe4L");
search.setQueryString(toSearch);
```

Oltre ad istanziare *GoogleSearch* bisogna settare la chiave che ci è stata inviata via email con il metodo *setKey()*. Poi possiamo anche settare un proxy, e relativa porta, attraverso il quale vogliamo effettuare la ricerca ed infine con il metodo *setQueryString()* settiamo la stringa che verrà ricercata su Google. Così abbiamo costruito la richiesta, ora dobbiamo collegarci al WebService e salvare i risultati (10 per default) in un vettore

```
Vector toReturn=new Vector();
GoogleSearchResult result = search.doSearch();
GoogleSearchResultElement[] re =
                          result.getResultElements();
for ( int i = 0; i < re.length; i++ ) {
    toReturn.add(re[i].getURL());
}
```

Ora tutti risultati della ricerca sono presenti all'interno del vettore. Possiamo quindi facilmente integrare questo servizio all'interno del nostro Bot. Prima definiamo un trigger nella solita maniera

```
else if (action.equals("!GOOGLE")) {
searchAction(st,sender,channel);
}
```

poi definiamo il metodo *searchAction()* richiamando questa classe che abbiamo implementato. Praticamente avremo un vettore come risultato quindi in questo metodo basterà eseguire un ciclo *for* e stampare nel canale tutti i risultati

```
public void searchAction(StringTokenizer st,String
                        sender,String channel) {
  try{
     String toSearch=st.nextToken();
     while(st.hasMoreTokens())
     toSearch=toSearch+" "+st.nextToken();
     WebSearch ws=new WebSearch(0);
     Vector result=ws.search(toSearch);
     sendMessage(channel,sender+" ecco i risultati
                                   della tua ricerca");
     for (int i=0;i<result.size();i++) {
         String temp=(String)result.elementAt(i);
         sendMessage(channel,temp); } }
  catch(Exception e) { }
}
```

Il risultato di questa estensione la possiamo ammirare nell'immagine qui a fianco. Allo stesso modo in cui abbiamo inserito la ricerca su Google possiamo inserire una ricerca avanzata su siti specializzati in articoli Java. Sfrutteremo la possibilità che offre Google di segnalare su quale sito effettuare una ricerca con l'opzione site:nomesito. Ecco il semplice metodo che riutilizza la normale ricerca

```
public Vector searchAdvanced(String toSearch){
  String toSearch1="site:www.javaworld.com
                                    "+toSearch;
  String toSearch2="site:www.javalobby.org "+toSearch;
  String toSearch3="site:www.alphaworks.ibm.com
                                    "+toSearch;
  tipo=0;
  Vector temp=search(toSearch1);
  temp=search(toSearch2);
  temp=search(toSearch3);
  return temp;
}
```

Sempre allo stesso modo definiremo un trigger e il metodo associato per parsare la richiesta e restituire i risultati.

```
* Now talking in #javastaff
<d0c> !GOOGLE java
<DeLiRiUm> d0c ecco i risultati della tua ricerca
<DeLiRiUm> http://java.sun.com/
<DeLiRiUm> http://java.sun.com/docs/books/tutorial/
<DeLiRiUm> http://www.java.com/en/download/windows_automatic.jsp
<DeLiRiUm> http://www.java.com/
<DeLiRiUm> http://www.anfyteam.com/java/
<DeLiRiUm> http://javaboutique.internet.com/
<DeLiRiUm> http://www.javaworld.com/
<DeLiRiUm> http://www.blackdown.org/java-linux.html
<DeLiRiUm> http://www.developer.com/java/
<DeLiRiUm> http://www.microsoft.com/mscorp/java/
```

Fig. 2: ***Risultato della ricerca su Google***

CONCLUSIONI

Nel codice allegato alla rivista trovate anche un'altra funzionalità, un semplice invio di email tramite il Bot. Come avete potuto vedere si possono realizzare tantissime cose e personalizzare in maniera esclusiva un proprio Bot. Nell'homepage di PircBot potete trovare tanti progetti che hanno scelto come punto di partenza questo framework ed hanno sviluppato dei Bot con delle interessantissime feature. Pircbot è un ottimo strumento per supportare la nostra creatività.

Federico Paparoni

Federico Paparoni, può essere contattato per suggerimenti o delucidazioni all'indirizzo email federico.paparoni@javastaff.com

Il pixel che ti cambia i connotati

Alla scoperta del Pixel Shader, per realizzare animazioni vicine alla qualità cinematografica, da utilizzare all'interno di videogames o per creare presentazioni spettacolari

REQUISITI

Conoscenze richieste
Basi di C++, DirectX

Software
Microsoft Visual C++, Direct 9.1

Impegno

Tempo di realizzazione

Le immagini che normalmente vediamo sono il risultato dell'incidenza dei raggi luminosi sulla retina, l'organo preposto alla trasformazione di questi stimoli in impulsi da inviare al cervello. I "raggi luminosi" sono onde elettromagnetiche che viaggiano in un certo campo di frequenze, detto, per evidenti motivi, "campo del visibile". Il cammino di queste onde comincia da una fonte luminosa, ad esempio il Sole o una normale lampadina; passa attraverso l'interazione con corpi solidi, che ne modificano alcune caratteristiche, come il colore da noi percepito e, infine, raggiunge i nostri occhi. Le regole di queste interazioni sono tutte abbastanza note nel campo scientifico e riproducibili attraverso computer.
Inutile dire che il sogno di ogni appassionato di computer grafica in tempo reale sarebbe quello di potere realizzare un algoritmo che calcoli "al volo" il risultato di queste interazioni. In questo modo si riuscirebbe ad ottenere una grafica assolutamente fotorealistica per giochi, presentazioni multimediali ecc. senza ricorrere ai vari "stratagemmi". Purtroppo i moderni computer non hanno ancora raggiunto l'efficienza di calcolo del cervello umano. Perciò ci dobbiamo accontentare di una simulazione ancora non del tutto credibile. Il problema principale sta, come si può intuire, nell'incredibile quantità di dati necessaria per questi calcoli. Quantità che rende praticamente impossibile, con le potenze di calcolo odierne, ottenere effetti "in tempo reale". Esistono però tecniche che consentono di avvicinarsi molto a questo obiettivo di realismo, e quasi tutte fanno uso dei *Pixel Shader*.

I PIXEL SHADER

Qualcuno di voi avrà sentito parlare di *Vertex Shader*. Per quelli che si sono persi il numero scorso di ioProgrammo, è bene ricordare che si tratta di una tecnica che consente di agire sui singoli vertici dei poligoni che approssimano una forma tridimensionale al fine di creare effetti spettacolari.
Il *Pixel Shader* si affianca al *Vertex Shader* e per certi versi si può ritenere ancora più preciso. Agisce infatti sul singolo Pixel che compone una scena tridimensionale al fine di stabilirne l'aspetto corretto e di simulare come il nostro occhio percepirebbe quel pixel se, anziché vederlo dentro un video, fosse parte di una scena reale. Per ogni pixel sul monitor, viene stabilito il colore in base a informazioni prese direttamente dalla scena 3D come ad esempio le caratteristiche del materiale che compone l'oggetto visualizzato, l'angolo formato dalla direzione dello sguardo e la sorgente luminosa ecc.
I *Pixel Shader* sono manipolabili attraverso veri e propri mini-programmi, che possono essere "caricati" ed "eseguiti" all'interno di codice scritto in linguaggi come C++ o VB. Il linguaggio che si utilizzava inizialmente per questi mini-programmi era una specie di assembler, decisamente poco leggibile.
Successivamente è stato introdotto un linguaggio molto più simile al C, che è presente in due versioni, sostanzialmente uguali tra loro. Si tratta di *Cg* di NVidia e *HLSL* di Microsoft.

I FILE AD EFFETTO

Programmare un *Pixel Shader* è una cosa abbastanza semplice, una volta che si comprende a fondo la logica di come agiscono.
Cominciamo descrivendo un semplice effect file, cioè il sorgente di un *Pixel Shader* che viene utilizzato all'interno di altri programmi.

```
// matrici passata dall'applicazione "ospite"
```

```
float4x4 matWorldViewProj: WORLDVIEWPROJECTION;
float4x4 matWorld  : WORLD;

// struttura per I dati di ciascun vertice
struct VS_OUTPUT
{
    float4 Pos : POSITION;
    float3 Light : TEXCOORD0;
    float3 Norm : TEXCOORD1;
};

// Vertex Shader – per ogni vertice riempie la relativa
// struttura
VS_OUTPUT VS(float4 Pos : POSITION, float3 Normal
                                          : NORMAL)
{
    VS_OUTPUT Out = (VS_OUTPUT)0;
    Out.Pos = mul(Pos, matWorldViewProj);
                               // trasforma la posizione
    Out.Light = normalize(vecLightDir);
                              // vettore raggio luminoso
    Out.Norm = -normalize(mul(Normal, matWorld));
                                             // normale
    return Out;
}

// Pixel Shader – Calcola il coloe di ciascun pixel in
// base al raggio luminoso e alla normale
// di ciascun vertice
float4 PS(float3 Light: TEXCOORD0, float3 Norm :
                                 TEXCOORD1) : COLOR
{
    // Intensita della luce ambientale
    float Aintensity=0.2f;

    // Colore della luce ambientale
    float4 Acolour=float4(0.1,0.1,0.1,1.0);

    // Intensita e colore variabili
    float Dintensity=1.0f;
    float4 Dcolour=float4(1.0,0.6,0.6,1.0);

    // Calcola il colore del pixel utilizzando il prodotto
    // scalare tra raggio luminoso e normale del vertice
    float4 result=Dintensity*Dcolour*(dot(Norm,Light));

    // Add the diffuse result to the ambient below for
                                               return
    return Aintensity*Acolour+result;
}

// Definisce la technique da usare
technique TVertexAndPixelShader
{
    pass P0 // singolo passo di elaborazione
    {
        VertexShader = compile vs_1_1 VS();
        PixelShader  = compile ps_1_1 PS();
    }
}
```

Questo codice permette di visualizzare una mesh come se fosse di un materiale opaco di colore arancione carico, simile al rame.
Analizziamo di seguito la struttura del file appena presentato.

ANATOMIA DI UN EFFECT FILE

I file *.fx* sono organizzati in maniera abbastanza rigorosa e ricordano la struttura di un programma C. Le tre sezioni principali sono le seguenti:

- *Dichiarazioni di variabili*
- *Funzioni*
- *Technique*

La prima sezione è abbastanza esplicativa: qui vengono dichiarate tutte le variabili o le strutture che saranno utilizzate nel codice seguente.
Ad esempio nel nostro caso con la riga:

```
float4x4 matWorldViewProj: WORLDVIEWPROJECTION;
```

stiamo istruendo il compilatore a considerare l'identificatore univoco *matWorldViewProj* come una matrice 4x4 di float *(float4x4)*. Il valore finale di questa variabile sarà utilizzato come matrice di proiezione tra le coordinate del mondo *(world)* e quelle della visuale *(view)*. Questo è specificato attraverso la parola chiave *WORLDVIEWPROJECTION*.
Le funzioni sono esattamente analoghe alle funzioni presenti in tutti i linguaggi di programmazione. Attraverso un nome identificativo è possibile svolgere una generica parte di codice, specificando i parametri di ingresso e quelli di uscita.
Nel nostro caso la funzione che esegue i calcoli per il *Pixel Shader* è definita come segue:

```
float4 PS(float3 Light: TEXCOORD0, float3 Norm :
                                 TEXCOORD1) : COLOR
{
    ...
}
```

Il valore restituito dalla funzione sarà un vettore di 4 float *(float4)* al quale sarà associato il colore *(COLOR)* del pixel per il quale la funzione viene eseguita. I parametri in ingresso, tra parentesi tonda, sono il vettore luminoso e la normale del vertice, entrambi vettori di 3 float *(float3)*. Questi sono passati utilizzando l'asso-

DOVE OSANO I PIXEL SHADER

I Pixel Shader entrano in gioco, nell'elaborazione dell'immagine 3D, dopo le trasformazioni geometriche e prima del disegno finale dell'immagine. La sequenza di tutti questi procedimenti è detta "*pipeline grafica*", mentre il disegno vero e proprio dell'immagine è detto "*rasterization*".
La rasterization viene effettuata come operazione finale e si occupa di applicare tutti gli effetti, anche quelli 2D come ad esempio l'*antialiasing*.

ciazione ai registri *TEXCOORD0* e *TEXCOORD1*, fatta nella dichiarazione.
Le sezione dedicata alle technique definisce il nome di una tecnica da utilizzare per applicare lo shader. All'interno di ciascuna tecnica è possibile specificare più di un passo di elaborazione, con la parola chiave *"pass"*.

LO SHADER

La funzione che effettua il calcolo vero e proprio del valore finale del colore del pixel è *PS()*.
Le righe:

```
float Aintensity=0.2f;
float4 Acolour=float4(0.1,0.1,0.1,1.0);
```

definiscono l'intensità e il colore della luce ambientale. È sempre bene tenerne conto per non fare risaltare troppo la mesh cui lo shader è applicato, rispetto allo "sfondo".
Successivamente con:

```
float Dintensity=1.0f;
float4 Dcolour=float4(1.0,0.6,0.6,1.0);
float4 result=Dintensity*Dcolour*(dot(Norm,Light));
```

si definiscono intensità e colore variabili del pixel e si stabilisce il colore finale moltiplicando questi valori per il prodotto scalare tra vettore luminoso e normale.

NOTA

GLI SHADER OGGI

Lo stato attuale della tecnologia consente l'utilizzo di shader che inglobano diverse caratteristiche dei linguaggi di programmazione comuni (ad esempio la presenza del costrutto "*if-then-else*"). La versione supportata dall'hardware di ultima generazione è la 3.0. Tuttavia le potenzialità degli shader sono ancora per la maggior parte da esplorare, lasciate alla creatività e capacità dei programmatori. C'è addirittura chi prevede in futuro una congiunzione tra *Vertex* e *Pixel Shader* in una sola entità.

CELL SHADING

In questo minitutorial realizzeremo qualcosa di molto vicino a un cartone animato. Seguiteci e i risultati saranno sorprendenti

> COSA È IL CELL SHADING?

1 Il "Cell Shading" è un effetto visivo che consente di disegnare la scena 3D come se fosse un cartone animato. Questa tecnica è stata resa celebre dal videogioco "XIII" e risulta di grande impatto.

> VIA CON LA PRIMA FORMULA

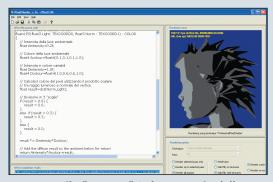
```
// Calcola il colore del pixel
// utilizzando il prodotto scalare
// tra raggio luminoso e normale
// del vertice

float result=dot(Norm,Light);
```

2 Possiamo modificare il nostro codice in modo da ottenere un effetto simile al Cell Shading. Per farlo agiamo sulla variabile "result" assegnandole semplicemente il solito prodotto scalare.

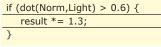
> EFFETTI DI LUCE

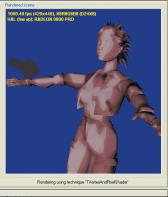
```
// Divisione in 3 "soglie"
if (result > 0.6) {
   result = 0.6;
}
else  if (result > 0.3) {
   result = 0.3;
}
else {
   result = 0.0;
}
result *= Dintensity*Dcolour;
```

3 Il segreto di questo effetto sta nel dividere in "soglie" i valori dell'incidenza del raggio luminoso. Successivamente si assegna a "result" un valore fisso, in base alla soglia in cui ricade.

> IL RISULTATO FINALE

4 Lo stile "cartoon" è dato proprio dalla presenza di campiture uniformi di colore, tipiche dei fumetti. L'effetto che si ottiene è quello di una figura disegnata a mano in bianco e nero.

COME PROVARE L'ESEMPIO

**Per provare lo shader potremmo scrivere un piccolo programmino con DirectX che carichi il codice e lo applichi a una mesh scelta da noi.
Una via molto più breve, descritta anche nel precedente articolo sui *Vertex Shader*, è quella di utilizzare *Effect Edit*, un tool presente nel DirectX 9.0 SDK scaricabile a partire dal sito** www.microsoft.com**.
Caricando questo shader in *EffectEdit* noteremmo subito il risultato dei nostri sforzi.
E, modificando "al volo" il codice che compare a schermo, potremmo subito verificare gli effetti di eventuali cambiamenti.
Tanto per prenderci gusto aggiungiamo nella funzione *PS()* le seguenti righe prima del *return*:**

```
if (dot(Norm,Light) > 0.6) {
    result *= 1.3;
}
```

**L'effetto viene ricompilato al volo da *Effect Edit* e il risultato che si può vedere è quello di una certa "lucidità" della mesh, rispetto a prima.
L'effetto è ottenuto aumentando il valore dell'intensità luminosa data da "*result*" di un 30%, nel caso in cui il prodotto tra *Norm* e *Light* sia maggiore di un certo valore di soglia (fissato a 0.6).
Questo è tipico dei materiali lucidi, ad esempio i metalli, che tendono a riflettere la luce quasi totalmente quando questa incide in maniera diretta. Da notare, in ogni caso, come un effetto completamente diverso sia stato ottenuto modificando minimamente il codice dello shader. Questo è un grande vantaggio dei *Pixel Shader.***

L'effetto viene ricompilato al volo da Effect Edit e il risultato che si può vedere è quello di una certa "lucidità" della mesh, rispetto a prima

Questo valore è tanto più alto quanto la luce "incide" sul pixel in maniera diretta.
L'effetto finale che si ottiene è proprio quello di un oggetto abbastanza realistico, di un materiale piuttosto opaco.
Da notare come oltre a *PS()* si è definita anche la funzione *VS()* che è un *Vertex Shader* (cioè un manipolatore della geometria delle mesh).
Questi tuttavia non fa altro che applicare le trasformazioni di default, per cui non compie nessun lavoro "utile".
Gli shader così definiti vengono applicati tramite le direttive *"compile"* specificate nella *techinque*.
In particolare le righe:

```
VertexShader = compile vs_1_1 VS();
PixelShader  = compile ps_1_1 PS();
```

compilano gli shader con le funzioni *PS()* e *VS()*, utilizzando le versioni 1.1 degli shader stessi.

CONCLUSIONI

Abbiamo visto in questo articolo come scrivere il codice di un semplice *Pixel Shader* e come provarlo immediatamente utilizzando il tool *Effect Edit* di DirectX. È facile intuire come piccole modifiche nei punti giusti dello shader possano portare a risultati di grande effetto, anche molto differenti tra loro.
Per una ulteriore modifica dello shader si legga anche il tutorial riportato in queste pagine.
I codici completi sono presenti in ogni caso sul CD allegato.

Alfredo Marroccelli

CARICARE UN FILE .FX CON DIRECTX 9

**I file *.fx* consentono di concentrare, in una sorta di script, praticamente tutte le operazioni richieste per inizializzare un effetto grafico.
Questo consente di eliminare molte parti di codice C++, evitando noiose e lunghe fasi di ricompilazione. Utilizzando DirectX 9 il caricamento di un effetto si riduce alla chiamata della funzione:**

```
HRESULT WINAPI
    D3DXCreateEffectFromFile(
    LPDIRECT3DDEVICE9 pDevice,
    LPCTSTR pSrcFile,
    const D3DXMACRO *pDefines,
    LPD3DXINCLUDE pInclude,
    DWORD Flags,
    LPD3DXEFFECTPOOL pPool,
    LPD3DXEFFECT *ppEffect,
    LPD3DXBUFFER
    *ppCompilationErrors
    );
```

***pDevice* è il puntatore al dispositivo *IDirect3DDevice9* che rappresenta la scheda grafica; *pSrcFile* è il nome del file che contiene l'effetto e *ppEffect* rappresenta l'effetto vero e proprio come oggetto C++.
Gli altri parametri sono principalmente opzioni di compilazione dell'effetto e possono essere lasciati a *NULL* oppure 0, per effetti semplici.
Per una guida più approfondita consigliamo direttamente il sito ufficiale** http://www.microsoft.com/windows/directx/default.aspx

Le immagini le faccio alla griglia

Sfruttiamo la potenza della programmazione ad oggetti per estendere le potenzialità del Datagrid control di .NET. Creiamo una galleria di thumbnail mostrando delle immagini nelle celle

Molti programmatori che si addentrano nello sterminato mondo del .NET Framework, magari provenendo dal buon vecchio VB6, si trovano disorientati nell'uso dei controlli standard forniti dal Framework e, molto spesso, rimangono meravigliati dalla mancanza di funzionalità che a prima vista sembrerebbero ovvie in un controllo che si rispetti. Molti casi di "disorientamento del programmatore" sono da attribuirsi proprio al famoso (o famigerato) controllo *Datagrid*, quello che viene comunemente utilizzato per mostrare in tabelle dei dati magari provenienti da un database. A prima vista infatti sembrerebbe che nelle celle della tabella di un Datagrid potrebbero trovar posto solo testo o al massimo dei miseri Checkbox.

Il fatto è che Microsoft ha seguito, nella progettazione dei controlli standard, il paradigma della programmazione ad oggetti, ovvero: è inutile introdurre centinaia di funzionalità in un controllo, tanto mancherà sempre quella che ci serve, meglio allora fornire un "semilavorato" personalizzabile a piacere. Questo approccio accresce senza dubbio la flessibilità. Il rovescio della medaglia è però che per sfruttare in pieno questa flessibilità occorre rimboccarsi le maniche per introdurre funzionalità "estendendo" gli oggetti base. Estremizzando proprio il concetto di estensione vedremo in questo articolo come utilizzare un controllo base, quale appunto la Datagrid, per fare da contenitore non a dati bensì ad immagini.

Il nostro obiettivo sarà quello di realizzare un programma in grado di visualizzare sotto forma di thumbnail le immagini contenute in una directory del file system.

Naturalmente la tecnica illustrata si presta a molte "variazioni sul tema" come, ad esempio, inserire delle icone nella griglia dei dati ecc...

IL CONTROLLO DATAGRID

System.Windows.Forms.DataGrid è un oggetto complesso che ha:

- un'origine dati (*Dataset, Datatable, Dataview, Matrice* ecc...);
- degli oggetti *DataGridTableStyle* che controllano l'aspetto delle griglie e appartengono alla collection *GridTableStylesCollection;*
- a loro volta ogni oggetto *DataGridTableStyle* ha una collection di oggetti che derivano dall'oggetto astratto *DataGridColumnStyle*.

Gli oggetti derivanti da *DataGridColumnStyle* che vengono comunemente impiegati nella griglia sono *DataGridTextBoxColumn* (che controlla le colonne con celle contenenti del testo) e *DataGridBoolColumn* (che controlla le colonne con valori *True/False* rappresentati con checkbox). Fin qui quello che "passa il convento" ovvero delle celle con textbox o checkbox, ma il bello è che nulla ci vieta (anzi!) di creare un nostro oggetto derivante da

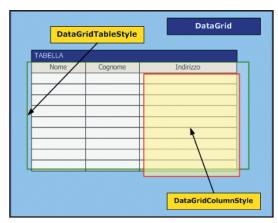



Fig. 1: *Schema componenti Datagrid*

DataGridColumnStyle per mettere nelle celle tutto quello che vogliamo. Per far questo abbandoniamo prima di tutto l'idea di lavorare in ambiente visuale: il tutto deve essere gestito scrivendo del codice, comunque vi accorgerete che l'operazione è più semplice di quanto possiate pensare.

UNO STILE PERSONALIZZATO

Mano al codice quindi, e andiamo a creare nel progetto una nuova classe, che chiameremo *CellExt* che eredita da *DataGridColumnStyle*:

```
Public Class CellExt
    Inherits DataGridColumnStyle
```

Per motivi di spazio rimandiamo l'analisi della struttura della classe *CellExt* al codice sorgente allegato nel CD, l'importante è aver chiaro che la nostra classe è chiamata in causa ogni volta che la fonte dei dati passa al *Datagrid* una path; in questo caso *CellExt* si occuperà di disegnare in quella determinata cella l'immagine che rappresenta il Thumbnail di quella originale. Il tutto sta nel metodo *Paint* che compie le seguenti operazioni:

1. Dato il numero di riga trova il valore corrispondente nella fonte di dati (ovvero la path di un'immagine).
2. Carica in memoria l'immagine.
3. Ridimensiona l'immagine proporzionalmente alla grandezza della cella (che nel nostro esempio abbiamo fissato in 100 X 100 pixel).
4. Disegna l'immagine nello spazio assegnato alla cella.

Il metodo si presenterà quindi grosso modo così:

```
Protected Overloads Overrides Sub Paint(ByVal g As
        System.Drawing.Graphics, ByVal bounds As
        System.Drawing.Rectangle, ByVal source As
        System.Windows.Forms.CurrencyManager, ByVal
        rowNum As Integer)
'Trova la path dato il numero di riga
Dim value = GetColumnValueAtRow(source, rowNum)
'disegna lo sfondo bianco della cella
g.FillRectangle(Brushes.White, bounds)
Dim thumb As Bitmap
'carica l'immagine
Dim bmp As New Bitmap(value.ToString)
Dim ResizeSize As Size = GetResizeSize(bmp)
'ridimensiona l'immagine alla grandezza della cella
thumb = bmp.GetThumbnailImage(ResizeSize.Width,
        ResizeSize.Height, Nothing, IntPtr.Zero)
Dim thumbX As Integer = bounds.X +
        ((bounds.Width - thumb.Width) / 2)
Dim thumbY As Integer = bounds.Y +
        ((bounds.Height - thumb.Height) / 2)
Dim r As New Rectangle(thumbX, thumbY,
        thumb.Width, thumb.Height)
'disegna l'immagine nel rettangolo
g.DrawImage(thumb, r)
End Sub
```

Precisiamo che la funzione *GetResizeSize* l'abbiamo definita noi per calcolare la riduzione proporzionale dell'immagine *Thumbnail* in relazione allo spazio a disposizione. La funzione *GetColumnValueAtRow* con cui invece troviamo la path del file viene invece gentilmente fornita da *DataGridColumnStyle* da cui la nostra classe eredita. Naturalmente nella pratica le cose sono un po' più articolate e nel codice sorgente di esempio presente nel CD abbiamo messo anche il *Caching* delle immagini per limitare i tempi di caricamento ed altre cose a corredo, comunque grosso modo le operazioni sono queste.

L'INTERFACCIA DEL PROGRAMMA

A questo punto restano da compiere altre due operazioni: creare una fonte di dati per la griglia e dirle che i dati devono essere gestiti dalla nostra classe *CellExt*. Cominciamo con il disegnare nel progetto una Form con tutto il necessario:

1. Una *Textbox* che ospita la *path* della directory da visualizzare.
2. Un *Button* che apre una finestra di dialogo che consenta all'utente di scegliere la directory.
3. Una *Datagrid*.

NOTA

DATAGRID E STILI

Il controllo *System.Windows.Forms.DataGrid* visualizza i dati in forma di griglia. La classe *DataGridTableStyle* rappresenta la griglia disegnata. Questo oggetto è da non confondere con la classe *DataTable* che può rappresentare una origine di dati per la griglia. La classe *DataGridTableStyle*, invece, rappresenta esclusivamente la griglia come disegnata nel controllo.

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

CLASSI E COMPONENTI NELL'AMBIENTE DI SVILUPPO

Creando in Visual Studio una classe che eredita da *DataGridColumnStyle* noterete nella finestra *Esplora soluzioni* l'icona che rappresenta il file è diversa da quella utilizzata comunemente per rappresentare le classi, questo avviene perché Visual Studio considera la classe come un componente visto che *DataGridColumnStyle* a sua volta eredita da *System.ComponentModel.Component.* L'implicazione pratica di tutto ciò è che tentando di aprire il file con il doppio clic l'ambiente di sviluppo genera un errore perché tenta di aprire una finestra di progettazione che per *DataGridColumnStyle* non è ammessa visto che il tipo è dichiarato come *Abstract.* Per aprire il file occorre invece selezionarlo in *Esplora soluzioni* e premere *F7* o l'icona visualizza codice.

OVERLOADS E OVERRIDES

Il metodo Paint è stato dichiarato con due parole chiave *Overloads* e *Override.* In pratica ciò significa che va a sostituire lo stesso metodo che è presente in *DataGridColumnStyle*, cioè prende il suo posto e modifica un comportamento predefinito.

Il controllo Datagrid viene diffusamente trattato nella library Microsoft all'indirizzo: http://msdn.microsoft.com/library/en-us/cpref/html/frlrfSystemWindowsFormsDataGridClassTopic.asp

Inseriremo poi, nella finestra di progettazione anche un componente *FolderBrowserDialog* che ci consentirà di disporre di un'interfaccia di selezione della cartella senza che l'utente debba digitarla nella textbox. La funzionalità del programma sarà quindi questa:

Fig. 2: ***Il componente FolderBrowserDialog***

1. sull'evento *Click* sul *Button* si apre la finestra di dialogo;
2. l'utente sceglie la directory;
3. si crea una *Datatable* contenente le Path dei file grafici contenuti nella directory prescelta;
4. si passa alla griglia la fonte dati appena creata dicendole che deve essere gestita dalla nostra classe *CellExt* anziché con i componenti *DataGridColumnStyle* standard.

Rimandiamo, anche qui, al codice sorgente contenuto nel CD di ioProgrammo per la maggior parte di questi passaggi.
Analizziamo però alcuni passi salienti.

LA FONTE DI DATI

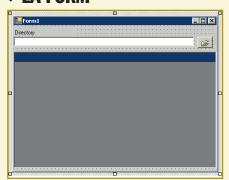
Questo è il codice che crea la *Datatable* a partire dalla directory prescelta:

```
'il percorso impostato nella TextBox
Dim d As String = TextBox1.Text
```

```
'crea la nuova DataTable assegnandogli un nome
Dim tb As New DataTable("fs")
'stabilisce il numero di colonne
Dim colCount As Integer = 3
For n As Integer = 0 To colCount - 1
    'crea le nuove colonne e le aggiunge alla
                                        DataTable
    Dim c As New DataColumn("file" & n,
                                GetType(String))
    c.ReadOnly = True
    tb.Columns.Add(c)
Next
'crea un arrayList con le path dei file grafici
Dim files As New ArrayList
If Directory.Exists(d) Then
    For Each f As String In Directory.GetFiles(d)
        Dim ext = Path.GetExtension(f)
                            .ToLower.TrimStart(".")
        Select Case ext
            Case "bmp", "jpg", "gif", "png"
                files.Add(f)
        End Select
    Next
End If
'crea un arrayList con le righe da inserire nella tabella
Dim rows As ArrayList = getRows(files, colCount)
For Each objs As Object() In rows
    'inserisce gli oggetti contenuti nell'arrayList "rows"
    'nella tabella
    tb.Rows.Add(objs)
Next
```

Fin qui nulla di particolare, salvo notare che sono state create 3 colonne per la tabella, ciò per mostrare una griglia di thumbnail di tre

UNA GRIGLIA IN SEI PASSI

> LA FORM

1 **In Visual Studio avviamo un progetto Visual Basic del tipo Windows Application e su una Form disegniamo un controllo textbox, un bottone e una datagrid.**

> LA CLASSE

```
Public Class CellExt
  Inherits DataGridColumnStyle
--
Protected Overloads Overrides Sub Paint(
        ByVal g As System.Drawing.Graphics,
           ByVal bounds As System.Drawing
        .Rectangle, ByVal source As System
  .Windows.Forms.CurrencyManager, ByVal
                      rowNum As Integer)
```

2 **Provvediamo a creare una nuova classe che eredita da DataGridColumnStyle. Quindi provvediamo ad estendere la nostra implementazione e sostituire il metodo Paint.**

> LA FONTE DATI

```
'crea un arrayList con le path dei file grafici
Dim files As New ArrayList
If Directory.Exists(d) Then
  For Each f As String In Directory.GetFiles(d)
  Dim ext = Path.GetExtension(f)
                      .ToLower.TrimStart(".")
  Select Case ext
     Case "bmp", "jpg", "gif", "png"
     files.Add(f)
  End Select
Next
    End If
```

3 **Otteniamo la fonte dei dati per la griglia partendo dalla directory selezionata dall'utente. Abbiamo reperito questa informazione utilizzando un folder BrowserDialog.**

celle per riga. Un elenco del tipo:

- *C:\documenti\file1.jpg*
- *C:\documenti\file2.jpg*
- *C:\documenti\file3.jpg*
- *C:\documenti\file4.jpg*

Verrà mostrato come:

C:\documenti \file1.jpg	C:\documenti \file2.jpg	C:\documenti \file3.jpg
C:\documenti \file4.jpg		

I GESTORI DEI DATI

Dobbiamo poi dire alla griglia con cosa dovrà gestire i dati che provengono dalla nostra *Datatable*. Per questo sviluppiamo una funzione che restituisce un gestore di tabella personalizzato:

```
Private Function getTs(ByVal tb As DataTable) As
                                        DataGridTableStyle
    'dichiara un nuovo gestore di tabella
    Dim ts As New DataGridTableStyle
    'associa il gestore di tabella alla Datatable
    ts.MappingName = tb.TableName
    'imposta varie proprietà di stile della tabella
    ts.ReadOnly = True
    ts.ColumnHeadersVisible = False
    ts.AllowSorting = False
    ts.RowHeadersVisible = False
    For Each c As DataColumn In tb.Columns
        'associa ad ogni colonna il gestore
        'personalizzato cellExt
        Dim cs As New CellExt
        cs.MappingName = c.ColumnName
        cs.ReadOnly = True
        cs.HeaderText = ""
        ts.GridColumnStyles.Add(cs)
    Next
    Return ts
End Function
```

Possiamo notare come la nostra classe *CellExt*, derivata da *DataGridColumnStyle,* entra in gioco mediante un'associazione operata a livello di ogni colonna della *DataTable* di partenza. Resta, a questo punto, soltanto da associare a sua volta il gestore di tabella alla *Datagrid* stessa, semplicemente con:

```
DataGrid1.TableStyles.Add(getTs(tb))
```

Dove naturalmente la variabile *tb* che passiamo alla funzione sarà la *Datatable* che abbiamo visto in precedenza.

CONCLUSIONI

Abbiamo visto come partendo da un controllo base come la *Datagrid* sia possibile, attraverso il meccanismo dell'estensione, ottenere funzionalità del tutto diverse dall'ordinario. Naturalmente quello illustrato è solo un esempio della tecnica dell'estensione; una volta acquisiti i concetti di base l'unico limite è la fantasia!

Francesco Smelzo

Francesco Smelzo è docente di Informatica presso l'ateneo di Siena ed è specializzato nello sviluppo in ambiente .NET. Si occupa anche del coordinamento dello sviluppo Software della Eurosoft Italia. È a disposizione per rispondere a quesiti e suggerimenti all'indirizzo francesco@smelzo.it

> CREIAMO LO STILE

```
Dim ts As New DataGridTableStyle
'associa il gestore di tabella alla Datatable
ts.MappingName = tb.TableName
'imposta varie proprietà di stile della tabella
...
For Each c As DataColumn In tb.Columns
'associa ad ogni colonna il gestore
'personalizzato cellExt
Dim cs As New CellExt
...
ts.GridColumnStyles.Add(cs)
Next
        Return ts
```

4 Impostiamo lo stile con cui verrà visualizzata la fonte dati da parte di Datagrid associando ad ogni colonna il gestore personalizzato che abbiamo creato in precedenza.

> PASSIAMO STILI E DATI

```
Private Function getTs(ByVal tb As
            DataTable) As DataGridTableStyle
'dichiara un nuovo gestore di tabella
'associa il gestore di tabella alla Datatable
'imposta varie proprietà di stile della tabella
'associa ad ogni colonna il gestore
'personalizzato cellExt
End Function

DataGrid1.TableStyles.Add(getTs(tb))
```

5 Dove getTs(tb) contiene i gestori di colonna controllati dalla classe che abbiamo visto al passo 2, mentre tb è la DataTable che abbiamo creato al passo 3. Impostiamo il DB al variare delle directory.

> IL RISULTATO FINALE

6 Il risultato sarà quindi una griglia che ospita i thumbnail delle immagini contenute nella directory selezionata. Esattamente quello che fa al caso nostro!

Paginazione su milioni di record in SQL Server

Costruiremo la pagina di ricerca di un sito e-commerce che può contare su di un'anagrafica di un milione di articoli. Vedremo come ottenere prestazioni da competizione grazie a SQL Server e Asp.NET

Paginare significa presentare i record restituiti da una query (*resultset*) in piccoli gruppi di elementi ordinati secondo un certo criterio. La paginazione è una tecnica utilizzata tipicamente sul web (motori di ricerca, siti di e-commerce) dove la trasmissione in blocco di migliaia di righe al client non sempre è possibile, agevole o opportuna.
Offrire resultset paginati permette di rendere meno difficoltosa agli utenti la consultazione dei risultati ottenuti. Dal punto di vista applicativo il problema della paginazione su SQL Server non ha ancora trovato una soluzione definitiva, soprattutto in presenza di grosse moli di dati. Neppure in questa sede sarà posta la parola fine al problema ma ugualmente verranno offerti alcuni spunti preziosi su cui lavorare.

I LAYER DELL'APPLICAZIONE

Consideriamo l'architettura di una classica applicazione web a tre livelli: livello dei dati, logica di business, livello di presentazione (architettura *Windows Distributed InterNet Applications* o *DNA*). Il problema che ci si pone nell'ottica della paginazione è: su che layer paginiamo? Il programmatore pigro deciderà di paginare nell'ambito della logica di business, magari utilizzando le capacità intrinseche di una datagrid, ottenendo risultati mediocri: tutto il resultset andrà infatti ad occupare la memoria del web server appesantendolo inutilmente, soprattutto nel caso in cui ogni utente acceda a grossi resultset personalizzati. Una buona soluzione è la paginazione al livello dei dati: in questo modo faremo lavorare molto di più il database, ma visto che il trattamento dei dati è il suo mestiere…

LO SCENARIO

L'idea è creare la pagina web di ricerca articoli di un sito di e-commerce. Il nostro sito di e-commerce si baserà su un catalogo di 1.000.000 di articoli. Un bel numero direi. Dalla ricerca, l'utente dovrà ottenere pagine suddivise in gruppi di 20 articoli ordinati per descrizione.
L'obiettivo finale è quello di ottenere tempi di risposta nell'ordine di qualche secondo anche su macchine non particolarmente dotate.
La tabella su cui lavoreremo avrà una forma del genere:

Articoli		
idArticolo	*int*	*4*
descrizioneArticolo	*varchar*	*250*
categoriaArticolo	*varchar*	*250*
prezzoArticolo	*decimal*	*9*
immagineArticolo	*image*	*16*

L'esempio è semplificato e la tabella andrebbe normalizzata, ma il caso è adatto al nostro scopo. Il campo su cui pagineremo, cioè il campo di ordinamento, sarà *descrizioneArticolo*.

COME INIZIARE

Chi non potesse sperimentare le tecniche di paginazione qui illustrate su SQL Server, non si scoraggi. Ecco come dotare la propria cassetta degli attrezzi di strumenti gratuiti e facilmente reperibili:

- **MSDE 2000 – è SQL Server 2000 con alcune limitazioni ma completamente gratuito. Fate attenzione all'installazione dove viene richiesta obbligatoriamente una password per l'utente amministratore.** http://www.microsoft.com /sql/msde/downloads /download.asp
- **Dbamgr2k – ottimo tool free per la gestione di MSDE 2000 creato dal nostro Andrea Montanari.** http://www.asql.biz/DbaMgr.shtm
- **Query commander – per chi volesse strafare, ecco un tool simile a Query Analizer con cui potrete scrivere le vostre query SQL con syntax highligthing e autocompletamento. Naturalmente free** http://querycommander .rockwolf.com/

REQUISITI

Conoscenze richieste
Principi di ASP.NET

Software
Visual Studio 2003

Impegno

Tempo di realizzazione

I SISTEMI DI PAGINAZIONE SU DATABASE

Esistono molte soluzioni di paginazione su database. Noi focalizzeremo l'attenzione sulla paginazione attraverso query SQL implementabili sia lato applicazione che lato dabatase sotto forma di store procedure.
La prima soluzione proponibile è la seguente:

```
SELECT TOP 20 * FROM Articoli WHERE idArticolo
                                          NOT IN
  (SELECT TOP 60 idArticolo FROM Articoli ORDER BY
                          descrizioneArticolo ASC)
ORDER BY descrizioneArticolo
```

La *query interna* seleziona i primi 60 record (3 pagine per 20 record) della tabella e ne torna le chiavi (*idArticolo*). La *query esterna* utilizza le chiavi precedenti per escludere dalla sua ricerca i primi 40 record e tornando i successivi 20. Quindi nell'esempio il risultato della query è la pagina numero 4 (record da 61 a 80).
Questo genere di soluzione comporta due tipi di problemi. Il primo è facilmente aggirabile: la query pagina solo su campi *IDENTITY* cioè dotati di chiavi univoche. Il secondo limite è invece più grave perché causa un decadimento notevole delle prestazioni: la query più esterna opera un filtro con *WHERE NOT IN* su un numero di *idArticolo* sempre più alto man mano che le pagine aumentano. Già alla pagina 4 del nostro esempio l'SQL si tradurrà in:

```
SELECT TOP 20 * FROM Articoli WHERE idArticolo
  NOT IN (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
                                    ……57,58,59,60)
```

Fate qualche prova e vedrete i tempi di risposta aumentare geometricamente. *WHERE NOT IN* non è certo il comando più agile da eseguire per un database di enormi dimensioni. La seguente soluzione è molto più interessante e offre maggiore flessibilità nel lavoro successivo di rifinitura.

```
SELECT * FROM Articoli WHERE idArticolo IN
  (SELECT TOP 20 * FROM
    (
      SELECT TOP 80 * FROM Articoli
      ORDER BY descrizioneArticolo ASC
    ) AS tab1
      ORDER BY tab1.descrizioneArticolo DESC
  ) AS tab2 ORDER BY tab2.descrizioneArticolo ASC
```

La *query interna* restituisce i primi 80 record della tabella ordinati in modo ascendente per descrizione articolo, cioè le prime 4 pagine. La seconda query utilizza *tab1* per estrarre i primi 20 record dello stesso resultset ordinato in senso discendente: vengono così estratti i record appartenenti alla pagina 4 cioè i record dall'80 al 61 *(DESC)*. La terza query, si occupa di rimettere nel giusto ordine (ascendente) i record ottenuti dalla query *blu* restituiti in *tab2*.
In buona sostanza il funzionamento di questa query si basa sugli ordinamenti prima ascendenti, poi discendenti e poi ancora ascendenti che 'spostano' al top del recordset i 20 record da restituire nella pagina. Il lato negativo di questa soluzione è, anche in questo caso, il decadimento precoce delle prestazioni (la crisi si manifesta già dopo alcune centinaia di pagine). Vediamo come ottenere un buon compromesso tra le due soluzioni e come aggirare i limiti evidenziati.

DIAMO UNA MANO A SQL SERVER

Il primo collo di bottiglia che incontriamo risiede nella query interna ed è senz'altro la clausola *ORDER BY.*

```
SELECT TOP 80 * FROM Articoli ORDER BY
                          descrizioneArticolo ASC
```

Paginazione vuol dire estrazione di gruppi di dati ordinati, da questo concetto non si può prescindere, e quindi qualunque query di paginazione soffrirà della sindrome di *ORDER BY.* Fortunatamente SQL Server ci offre la possibilità di aggiungere un *indice cluster* alla nostra tabella.

MYSQL VS SQL SERVER

È importante sottolineare che in mySQL esiste un'istruzione del tipo *SELECT * FROM TABELLA LIMIT 40,10* che restituisce 10 record a partire dal 41, e che risolverebbe in un colpo solo tutti i nostri problemi. In SQL Server almeno fino alla versione attuale questa clausola non è disponibile e dobbiamo accontentarci cella clausola top.
Resta da vedere quanto l'istruzione *limit* sia performante in MySQL, tuttavia è comunque innegabile almeno la comodità dell'esistenza di questo costrutto.

Per capire cosa sia un *indice cluster* immaginiamo la disposizione fisica dei dati della tabella *Articoli.*
Hanno certamente un ordine "naturale" che sarà l'ordine di inserimento dei dati. Creando un indice cluster sulla colonna *descrizioneArticolo* diremo a SQL Server che l'ordine dei record non dovrà più essere quello di inserimento ma l'ordine alfabetico (ascendente nel nostro caso) della colonna clusterizzata che quindi potrà essere ordinata con *ORDER BY* in modo molto veloce. L'indicizzazione avrà effetto anche per gli inserimenti di nuovi record. In realtà il meccanismo adottato da SQL Server per la creazione degli indici non è così semplice ma tanto basta sapere per i nostri scopi. L'indice, grazie a

DBAMGR2K E LE QUERY

Per lanciare una query in *dbamgr2k* è necessario attivare lo strumento *Query* disponibile nel menu *Activity.* Con le *Query* si possono caricare file *.sql* già pronti o editare le istruzioni SQL direttamente nell'editor.

dbamgr2k, può essere creato attraverso il menu contestuale della tabella *Articoli* oppure attraverso l'istruzione:

```
CREATE CLUSTERED INDEX idSuDescrizioneArticolo
                    ON dbo.Articoli (descrizioneArticolo)
```

I due procedimenti sono esattamente equivalenti. Naturalmente, per sua stessa natura, ad una tabella può essere associato un solo *indice cluster.* Ma nulla ci vieta di creare altri indici (non cluster) per ognuno dei campi che vorremo abilitare all'ordinamento e quindi alla paginazione.

AUTENTICAZIONE MISTA E INTEGRATA

SQL Server permette l'accesso ai suoi database attraverso la gestione delle utenze interna oppure delegando il controllo delle stesse a Windows.
Il primo tipo di autenticazione è sempre disponibile mentre il secondo va attivato attraverso il menu di connessione di *Dbamgr2k* spuntando la voce *Trusted NT Connection.* Il menu di connessione si ottiene con un doppio click sull'iconcina di SQL Server

OTTIMIZZIAMO LA QUERY

Torniamo ora a testare la nostra query principale estraendo un numero di pagina più alto.

```
SELECT TOP 400000 * FROM Articoli ORDER BY
                          descrizioneArticolo ASC
```

Estraendo pagine molto alte, in questo caso la pagina 20.000 (400.000/20), vi accorgete che le prestazioni, grazie agli indici, migliorano di molto ma non raggiungono ancora risultati soddisfacenti.
Consideriamo che nel tracciato record di *Articoli* ci sono campi molto pesanti da restituire. Nella fattispecie i campi *descrizioneArticolo* e *categoriaArticolo* sono dei *varchar* da 250 caratteri e il campo *immagineArticolo* è addirittura un campo image contenente le immagini dei nostri prodotti (blob).
Nell'esempio che segue

```
SELECT idArticolo, descrizioneArticolo FROM
  (SELECT TOP 20 idArticolo, descrizioneArticolo FROM
    (
      SELECT TOP 400000 idArticolo,
                   descrizioneArticolo FROM Articoli
      ORDER BY descrizioneArticolo ASC
    ) AS tab1
      ORDER BY tab1.descrizioneArticolo DESC
  ) AS tab2 ORDER BY tab2.descrizioneArticolo ASC
```

la nostra query si limita ad estrarre gli unici due campi indispensabili alla paginazione: il *campo indice (idArticolo)* e il *campo di ordinamento (descrizioneArticolo)* ottenendo finalmente delle prestazioni accettabili. Purtroppo la nostra pagina di ricerca ha bisogno di un resultset completo per visualizzare tutti i dati relativi agli articoli della pagina. Come risolvere il problema?
Consideriamo il seguente esempio:

```
SELECT * FROM Articoli WHERE idArticolo IN
  (SELECT TOP 20 tab1.idArticolo FROM
    (
      SELECT TOP 400000 idArticolo,
                   descrizioneArticolo FROM Articoli
      ORDER BY descrizioneArticolo ASC
    ) AS tab1
      ORDER BY tab1.descrizioneArticolo DESC
  )
```

La nostra query ora è completa. La query SQL più interno si occupa di restituire gli *idArticolo* che appartengono alla pagina. La query più esterna (la *WHERE IN*) ci restituisce invece l'intero resultset che verrà utilizzato dalla nostra pagina di ricerca. Perché la query sia performante va indicizzata anche la chiave di paginazione (*idArticolo*) oltre che il campo di ordinamento (*descrizioneArticolo*). La query più esterna sfrutta gli *idArticolo* restituiti dalle query più interne per estrarre il resultset completo attraverso la *WHERE IN.* Notiamo anche che *descrizioneArticolo*, restituito dalla query interna *(rossa)* viene solo utilizzato per l'ordinamento dalla query intermedia *(blu)* ma non più restituito alla vera query di estrazione *(nera).* La pesantezza del comando *WHERE IN* è molto mitigata dal fatto che questo agirà solo su 20 *idArticolo* a prescindere dal numero di pagina estratto. Consiglio di creare un indice anche sul campo di paginazione *(idArticolo)* oltre all'indice cluster già creato.

```
CREATE UNIQUE INDEX idSuIdArticolo ON dbo.Articoli
                                      (idArticolo)
```

Se in un caso reale non aveste una chiave univoca, l'introduzione della *WHERE IN* produrrebbe resultset non corretti. Per ovviare al problema potete creare un *id* univoco da utilizzare solamente come chiave di paginazione:

```
ALTER TABLE Articoli ADD id int IDENTITY
```

Attenzione ad un altro problema: l'*ORDER BY* della query intermedia inverte l'ordinamento dei campi *descrizioneArticolo* (clausola *DESC*) basandosi solo sul loro ordine alfabetico e non più sull'ordine naturale dei record (ricordate il funzionamento dell'*indice cluster*?). Siccome non abbiamo la sicurezza che *descrizioneArticolo* sia univoco, due o più articoli potrebbero avere la stessa descrizione e quindi dobbiamo garantirci da ordinamenti non corretti. Lo stesso discorso vale per la

BIBLIOTECA

- **PROGRAMMARE VISUAL BASIC.NET** ***F. Balena***
- **C# GUIDA PER LO SVILUPPATORE** ***S.Robinson***
- **MICROSOFT SQL SERVER 2000 SYSTEM ADMINISTRATION, (Exam) 70-228**

query più esterna. Il risultato di queste ultime considerazioni può essere quello che segue:

```
SELECT * FROM Articoli WHERE idArticolo IN
   (SELECT TOP 20 tab1.idArticolo FROM
      (
         SELECT TOP 400000 idArticolo,
                   descrizioneArticolo FROM Articoli
         ORDER BY descrizioneArticolo ASC
      ) AS tab1
         ORDER BY tab1.descrizioneArticolo DESC,
                              tab1.idArticolo DESC)
ORDER BY descrizioneArticolo ASC,idArticolo ASC
```

Aggiungendo i due doppi ordinamenti (in prima istanza per *descrizioneArticolo* e in seconda istanza per *idArticolo*) ci preserviamo da ordinamenti scorretti dovuti alla non univocità del campo *descrizioneArticolo*. Il campo *idArticolo* essendo un'identità garantirà sempre l'esistenza di un ordinamento.

FINALMENTE LA PAGINA DI RICERCA

L'applicazione che mette in atto la nostra strategia di ricerca si compone di due semplici pagine *aspx: WebFormRicerca.aspx* e *GetImg.aspx*. La prima contiene la logica di paginazione, la seconda renderizza nella prima le immagini contenute nel campo *immagineArticolo* se presenti. Passiamo a commentare il codice C# della pagina *WebFormRicerca.aspx*. La stringa di connessione presuppone l'accesso a SQL Server in modalità di autenticazione mista; va parametrizzata con il nome del server in cui risiede il vostro SQL *Server (Data Source)*, il nome del database *(Initial Catalog)*, l'utente di connessione *(User Id)* e relativa password *(Password)*.

```
SqlConnection sqlConnection1 = new
            SqlConnection("Data Source=vmcomp;
                       Initial Catalog=1000000;
                       User Id=sa;Password=;
                       Connect Timeout=100");
```

Nel caso voleste accedere a SQL Server in modalità di autenticazione integrata, la specificazione dell'utente e della password lasciano il posto alla voce Integrated *Security=SSPI;*. La query che viene passata al *SqlCommand1* può contenere il parametro di ricerca (*WHERE descrizioneArticolo LIKE@descrizioneArticolo*) per permettere effettivamente di filtrare gli articoli che rispondono ad una determinata descrizione.

La query così com'è implementata ha però un difetto: pagina all'infinito. Gli articoli dell'ultima pagina vengono ripresentati anche in quella successiva e così via. Il motivo risiede nel fatto che quando si richiede ad una *SELECT* di estrarre i primi 1000 record (TOP 1000), se il resultset è costituito di 800 record, non ci sarà modo di sapere che mancano 200 record. Per ovviare al problema va fatto un preventivo conteggio dei record costituenti il recordset. Dal numero di articoli così ottenuti va calcolato il limite massimo di pagine ottenibili dalla query.

CREARE UN DB DA UN MILIONE DI RECORD

Nel cd allegato potrete trovare una store procedure che lanciata in SQL Server permetterà di creare il database ShopOnLine con la tabella Articoli. Il contenuto dei campi è casuale e la tabella è sprovvista delle immagini. Per diminuire il tempo di attesa necessario alla creazione della maxi tabella si può limitare il numero di record intervenendo sulla riga:

```
SET @NumeroRecord = 1000000
```

I blob delle immagini estratti dai campi *immagineArticolo* vengono riposti in un hashtable esposto dalla proprietà *static hashTableImmagini;* servirà alla pagina *GetImg.aspx* per estrarre le immagini senza dover interrogare nuovamente il database. Per presentare i risultati si è utilizzato un *Repeater* costituito di tutti i campi del database.

CONCLUSIONI

A questo punto la domanda potrebbe essere: la soluzione può essere implementata con successo in un ambiente di produzione?
L'esperienza insegna che ogni caso reale merita una attenta analisi, considerazioni specifiche e soluzioni ad hoc.
Posso anticipare qualcuna delle vostre questioni dicendo che la soluzione si comporta molto bene anche con query in join e con filtri *WHERE* multipli ma va lavorata e rivista ad ogni modifica e soprattutto vanno usati correttamente gli indici sulle varie tabelle coinvolte.
Una considerazione che finora ho accuratamente evitato ma che è forse la più importante riguarda la potenza dell'hardware che è il vero fattore fondamentale della paginazione. Un db server dovrebbe sempre essere ben carrozzato e molto veloce per garantire risposte in tempi accettabili soprattutto in ambito web.
Non bisogna poi dimenticare che una soluzione di paginazione su SQL Server può appoggiarsi a store procedure, che garantiscono prestazioni anche migliori ma una complessità nella costruzione più elevata.

Gianluca Negrelli

SUL WEB

http://www.codeproject.com/aspnet/PagingLarge.asp

http://lorenzobraidi.blogspot.com/2004/12/gestire-la-paginazione-dei-dati.html

http://www.aspfaq.com/show.asp?id=2120

http://www.devleap.com/SchedaArticolo.aspx?IdArticolo=10690

http://www.aspitalia.com/articoli/dna.aspx

AUTORI

Gianluca Negrelli si occupa di analisi architetturale e programmazione di applicazioni enterprise winform e web con framework .NET sia in C# che in VB. Si occupa di sistemi di accesso e gestione dati di database in ambiente Windows.
gianluca.negrelli@gmail.com

Un player audio in visual Basic

Vi guideremo all'uso del controllo "media control interface" il cuore pulsante di VB per la gestione di suoni e filmati. Infine realizzeremo un'applicazione completa per la riproduzione di CD ed MP3

In Visual Basic per amministrare le periferiche ed i file multimediali si possono utilizzare diverse tecnologie, in questo articolo mostreremo come farlo con gli strumenti base quali il controllo MCI e la funzione API *MCISendString*.
I device audiovisivi controllabili con l'MCI *(Media Control Interface)* spaziano dal lettore CD/DVD al videoregistratore. In questa puntata ci occuperemo soltanto del lettore CD e dell'esecuzione di file sonori quali *Wav, Mp3, Wma* e *Mid;* nel successivo appuntamento, invece, continueremo l'esplorazione delle caratteristiche multimediali di Windows occupandoci tra l'altro dei file video e della registrazione di file sonori e video.
Ricordiamo che i file con estensione *Wav* (formato *wave*) non sono altro che la pura registrazione in digitale dei suoni reali, non sono cioè compressi, essi sono supportati in Windows già dalla versione 3.1. Dato che i file *Wav* occupano molto spazio disco è quasi impossibile utilizzarli nelle comunicazioni. Per questo scopo, invece, sono adatti i file in formato *Mp3* e *Wma (Windows Media Audio)* ricavati con degli algoritmi di compressione *(CoDec)* che riducono le dimensioni dei file e contemporaneamente salvaguardano la qualità dell'audio.

API E CONTROLLO MCI

Il controllo *MCI* e la funzione *MCISendString*, possono pilotare le periferiche audiovisive attraverso dei semplici comandi in formato stringa, questi nel caso del controllo *MCI* sono anche associati ai suoi pulsanti; infatti, il controllo *Multimedia MCI* quando è trascinato su un form si presenta come un pannello con nove pulsanti, di default disabilitati.

Il controllo *MCI* è contenuto nella libreria *Microsoft Multimedia Control 6.0,* di default è nominato *MMControl1*.

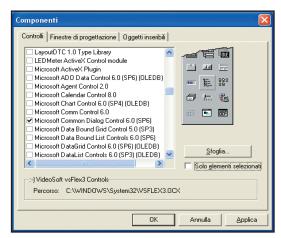



Fig. 1: *Il controllo è contenuto nella libreria Microsoft Multimedia control 6.0*

Per selezionare il device (cioè la periferica) con il quale si vuole interagire, il controllo *MCI* fornisce la proprietà *DeviceType*. I tipi di device supportati dal controllo sono: *CDAudio, WaveAudio, MPEGVideo, Sequencer, Other, AVIVideo, DAT, DigitalVideo, MMMovie, Overlay, Scanner, VCR* o *Videodisc*. S'intuisce che per controllare un lettore CD bisogna utilizzare il device *"CDAudio"*. Viceversa, per eseguire un file *Mp3* o *Wma* è necessario usare il *Device MPEGVideo*, dato che l'*Mp3* nasce dal *MPEG*.
Per inviare, un comando base, al device, a livello di programmazione, bisogna utilizzare la proprietà *Command*, che ha la seguente sintassi:

```
MMControl1.Command = "Comando"
```

Dove la stringa *Comando* può essere impostata con uno dei comandi base presentati nella **Tabella1**.

LE PROPRIETÀ BASE

Gli altri elementi notevoli dell'*MMControl*, sono le proprietà *Wait, AutoEnable, TimeFormat, Track, TrackLength, TrackPosition, From, UpdateInterval* e gli eventi *StatusUpdate, ButtonClick*. La proprietà *Wait* determina se il controllo deve attendere la fine di un comando prima di eseguire il successivo. *AutoEnable* stabilisce se il controllo deve attivare i propri pulsanti automaticamente in base al tipo di periferica. La proprietà *TimeFormat* specifica, il formato di ora utilizzato per la posizione *(position)* di esecuzione del brano. Per questa proprietà sono disponibili vari formati, nel nostro esempio abbiamo utilizzato il valore *mciFormatMilliseconds* (millisecondi).
Con la proprietà *UpdateInterval*, invece, si stabilisce quanti millisecondi devono intercorrere tra due eventi *StatusUpdate;* evento, quest'ultimo, utilizzato per recuperare in tempo reale i dati sull'esecuzione dei brani musicali. Le proprietà *Track, TrackLength* e *TrackPosition*, invece, forniscono informazioni sul numero e lunghezza della traccia, in esecuzione, e sulla posizione d'inizio della traccia. La proprietà *From* si utilizza per specificare la posizione d'inizio dell'esecuzione del prossimo comando *Play* o *Record*. Infine gli eventi *ButtonClick* (dove button è il nome di un pulsante, per esempio *Play_Click*) sono generati, quando l'utente clicca un pulsante del controllo; in realtà ai pulsanti sono associati altri eventi e proprietà come *ButtonCompleted, ButtonGotFocus, ButtonEnabled* ecc.

LE FUNZIONI API

Per la gestione dei file Mp3 è utile descrivere brevemente le seguenti funzioni API: *mciSendString, GetShortPathName* e *mciGetErrorString*. Più avanti sarà chiaro il ruolo della *GetShortPathName*, che serve per recuperare il path corto dei file e della *mciGetErrorString* che restituisce la descrizione degli errori *MCI*. Le dichiarazioni dettagliate delle funzioni si trovano nel progetto inserito nel CD, allegato alla rivista. La *mciSendString*, come accennato, guida le periferiche attraverso l'invio di comandi. La sintassi della funzione è la seguente:

```
mciSendString (comando, ReturnString,
                              ReturnLength, hwnd)
```

Il primo parametro è la stringa di comando da inviare alla periferica, il secondo è il buffer che conterrà i dati restituiti dalla funzione, il terzo è la lunghezza del buffer, l'ultimo parametro è l'handle della finestra che riceverà i messaggi di notifica. Di solito gli ultimi due parametri si utilizzano, quando si chiedono informazioni sullo stato della periferica (posizione, numero traccia ecc.). I comandi che è possibile inviare con la *mciSendString* sono molto complessi, per questo preferiamo schematizzarli con il seguente modello:

```
mciSendString ("ComandoBase & Device or Alias &
                                   altrivalori", , ,)
```

	Pulsante	Descrizione
Open	*Non previsto*	Apre un device
Close	*Non previsto*	Chiude un device
Play	*Play*	Esegue un file sonoro
Pause	*Pausa*	Sospende la riproduzione o la registrazione
Stop	*Stop*	Interrompe la riproduzione o la registrazione
Seek 85 to	*Dietro*	Passa all'inizio della traccia corrente o della precedente
Seek 85 to	*Avanti*	Passa all'inizio della traccia successiva
Record	*Record*	Registra
Eject	*Espelli*	Espelle il supporto della periferica
Save	*Non previsto*	Salva il file registrato
Set	*Non previsto*	Imposta diversi parametri, per esempio il formato ora
Status	*Non previsto*	Restituisce informazioni sull'esecuzione (traccia, lunghezza, posizione, 85)

Tabella 1: Comandi base per mciSendString

I comandi base, principali, sono presentati nella **Tabella 1**. Device può essere un nome di file o un tipo di periferica, a tal proposito si controlli la proprietà *DeviceType* del controllo *MCI*. Alias è un nome alternativo per individuare il device. *Altrivalori* sono dei parametri specifici dipendenti dal comando base.
Per esempio per aprire un file *Mp3* possiamo utilizzare il seguente comando:

```
Err = mciSendString ("OPEN " & filename & " TYPE "
                    & tipo & " ALIAS " & mioalias, 0, 0, 0)
```

Dove *OPEN* è il comando base, *filename* è il nome del file *Mp3* da aprire, tipo è il tipo di *Device* e *mioalias* è l'alias che si dà al file. *Err*, invece, è il valore restituito dalla funzione, questo è uguale a zero se il comando è eseguito correttamente, altrimenti è uguale al numero di errore generato.
Per conoscere la descrizione dell'errore bisogna passare il valore di *Err* alla *mciGetErrorString*, come mostreremo negli esempi. Facciamo notare che il path del file (cioè filename) deve essere quello corto, cioè del tipo *"C:\DOCUME~1\mioute\DOCUME~1\Musica\esemp.MP3"* altrimenti il comando non viene eseguito correttamente, questo giustifica la

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

necessità della *GetShortPathName*. Dopo aver aperto il file per eseguirlo si può utilizzare un'istruzione come la seguente:

```
mciSendString "PLAY " & mioalias & " from " & pos,
                                           0, 0, 0
```

Il *from*, di solito, viene utilizzato quando si deve eseguire un brano a partire da una data posizione, (specificata con *pos*), quindi non è sempre necessario. Per chiudere il device aperto, invece, si può usare la seguente:

```
mciSendString "CLOSE " & mioalias, 0, 0, 0
```

DALLA TEORIA ALLA PRATICA

Nei paragrafi precedenti abbiamo descritto in linea teorica il controllo *MCI*, in questo paragrafo presentiamo un lettore di CD audio che utilizza quanto abbiamo detto fin qui. Il lettore oltre a permettere la selezione dei brani musicali, visualizza alcune informazioni sullo stato dell'esecuzione e consente di spostare (in avanti o indietro) la posizione di lettura corrente. Quest'ultimo verrà gestito con un controllo *Slider*, o dispositivo di scorrimento, uno strumento che permette di associare lo scorrimento o il trascinamento di un cursore, su una barra graduata, ai valori di una determinata proprietà.

GLOSSARIO

FREQUENZA DI CAMPIONAMENTO

Rappresenta il numero di campioni di segnale sonoro presi per secondo, si misura in KHz, per esempio una frequenza di campionamento comune per file musicali è 44.1 KHz.

1 Create un nuovo progetto e referenziate le librerie: *Microsoft Multimedia Control 6.0* e *Microsoft Common Controls*, quest'ultima contiene il controllo *Slider*. Sul *form1* disponete un controllo *MCI*, delle Label e un controllo *Slider* come mostrato in **Figura 2**.
Le label servono per visualizzare il numero di traccia *(Labeltraccia)*, la durata in minuti della traccia *(LabelTot)* e il numero di minuti rimasti alla fine dell'esecuzione *(Labelposizione)*.

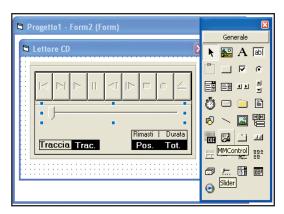
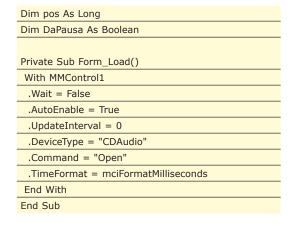
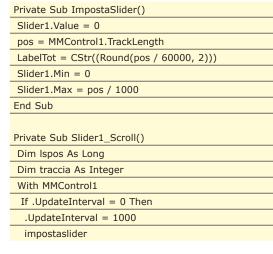



Fig. 2: *Il lettore CD in fase di progettazione*

2 Inizializziamo le dichiarazioni globali e definiamo la *Form_Load*.

```
Dim pos As Long
Dim DaPausa As Boolean

Private Sub Form_Load()
 With MMControl1
  .Wait = False
  .AutoEnable = True
  .UpdateInterval = 0
  .DeviceType = "CDAudio"
  .Command = "Open"
  .TimeFormat = mciFormatMilliseconds
 End With
End Sub
```

La variabile pos contiene la posizione corrente, in millisecondi, durante l'esecuzione del brano. *DaPause*, invece, è una variabile che indica se è stato premuto il tasto *Pause*. Nella *Form_Load* s'impostano le proprietà principali del controllo *MCI* e si apre la periferica *CDAudio* con il comando *Open*. Se nel lettore c'è un CD audio, eseguendo la *Form_Load*, si abilitano i pulsanti: *Play*, *Next*, *Prev* ed *Eject*. Se, invece, il lettore è vuoto si abilita solo il pulsante *Eject*.

3 Lo *Slider*, come accennato, viene utilizzato per controllare la posizione del lettore istante per istante. A tal fine nella procedura *ImpostaSlider* inseriamo le istruzioni che impostano le proprietà *Min*, *Max*. La proprietà *Value*, dello *Slider*, invece, viene incrementata ogni secondo nell'evento *MMControl1_Status Update*, così il cursore del controllo si sposterà man mano che il brano viene eseguito. Dello *Slider*, inoltre, utilizzeremo l'evento *Slider1_Scroll*, generato, quando si trascina, con il Mouse o la con la tastiera, il suo cursore.

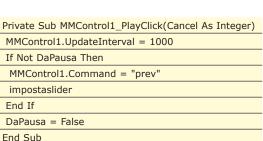
```
Private Sub ImpostaSlider()
 Slider1.Value = 0
 pos = MMControl1.TrackLength
 LabelTot = CStr((Round(pos / 60000, 2)))
 Slider1.Min = 0
 Slider1.Max = pos / 1000
End Sub

Private Sub Slider1_Scroll()
 Dim lspos As Long
 Dim traccia As Integer
 With MMControl1
  If .UpdateInterval = 0 Then
   .UpdateInterval = 1000
   impostaslider
```

```
End If
lspos = Slider1.Value
traccia = .Track
.Command = "close"
.Command = "open"
.TimeFormat = mciFormatMilliseconds
.Track = traccia
.From = .TrackPosition + lspos * 1000
Labelposizione = " " _
& CStr((Round((lspos * 1000) / 60000, 2)))
pos = (Me.Slider1.Max - lspos) * 1000
.Command = "play"
End With
End Sub
```

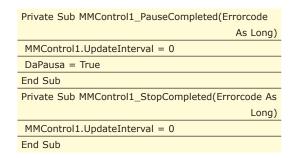
Nella *ImpostaSlider* le proprietà *Max* è impostata in base alla lunghezza della traccia che si sta eseguendo, essa è pari alla lunghezza in millisecondi della traccia diviso 1000 (1 sec =1000 m sec). Viene anche impostata *LabelTot* con la lunghezza in minuti della traccia (notare che 1 min = 60000 m sec). Nella *Slider1_Scroll*, invece, utilizzando la proprietà *From*, s'imposta la posizione da dove eseguire il comando *Play*. La posizione da impostare con il *From* è data dalla posizione iniziale della traccia (cioè *TrackPosition*) più la posizione impostata sullo *Slider*. La *TrackPosition* è necessaria dato che la posizione fornita da *lpos* è quella relativa rispetto all'inizio della traccia. La posizione del lettore, ricavata in base allo *Slider*, è anche utilizzata per impostare la *Labelposizione* e la variabile globale *pos*. Nell'evento *MMControl1_StatusUpdate* come accennato inseriamo il codice da eseguire man mano che il brano musicale viene riprodotto.

```
Private Sub MMControl1_StatusUpdate()
Labeltraccia = " " + CStr(MMControl1.Track)
Slider1.Value = Slider1.Value + 1
pos = pos - 1000
If CStr((Round(pos / 60000, 2))) < 0 Then
MMControl1.Command = "prev"
impostaslider
End If
Labelposizione = " " _
& CStr((Round(pos / 60000, 2)))
End Sub
```

In particolare in *MMControl1_StatusUpdate* incrementiamo il valore *(value)* dello *Slider*, decrementiamo la variabile *pos* (inizialmente impostata sulla lunghezza della traccia) ed impostiamo, con i valori correnti, la *LabelTraccia* e la *LabelPosizione*. Inoltre, in base al valore della variabile *pos* controlliamo quando la traccia in esecuzione termina (cioè quando arriviamo ad avere valori negativi per *pos*). Questo controllo è necessario dato che, anche se il lettore passa all'esecuzione del brano successivo il numero di traccia non viene aggiornato automaticamente. Quando si raggiunge la fine della traccia vengono reimpostati i valori dello *Slider* e delle *Label*, questo si fa con l'esecuzione del comando *"Prev"*.

4 Rimangono da implementare i comandi *"play"*, *"prossima traccia"*, *"traccia precedente"*. Facciamo notare che se il comando *"prev"* è eseguito una sola volta il controllo non passa alla traccia precedente ma all'inizio di quella corrente.
Per passare alla traccia precedente il comando deve essere eseguito due volte entro tre secondi. Un altro evento utilizzato è *PlayClick*.

```
Private Sub MMControl1_PlayClick(Cancel As Integer)
MMControl1.UpdateInterval = 1000
If Not DaPausa Then
MMControl1.Command = "prev"
impostaslider
End If
DaPausa = False
End Sub
```

Nell'evento *PlayClick* oltre ad impostare l'*UpdateInterval* si stabilisce, in base al valore di *DaPausa*, da quale posizione iniziare l'esecuzione. Gli eventi legati ai pulsanti *Pause* e *Stop*, invece, bloccano l'esecuzione del brano musicale (e quindi *StatusUpDate*).

```
Private Sub MMControl1_PauseCompleted(Errorcode
As Long)
MMControl1.UpdateInterval = 0
DaPausa = True
End Sub
Private Sub MMControl1_StopCompleted(Errorcode As
Long)
MMControl1.UpdateInterval = 0
End Sub
```

Negli eventi *Prev* e *Next*, invece, basta soltanto reimpostare lo *Slider*.

```
Private Sub MMControl1_PrevCompleted(Errorcode As
Long)
impostaslider
```

I canali possono essere Stereo o mono, il numero di bit può essere 8, 16 ecc. Ad esempio per una conversazione telefonica sono sufficienti campioni a 8 bit, campionati a 8 kHz mentre per un brano musicale è necessario un campionamento stereo a 16 bit con una frequenza di 44,1 kHz. Per quanto riguarda il BitRate, maggiore è il suo valore, migliore è la qualità del suono, però, di conseguenza maggiore è lo spazio di memoria occupato.

BITRATE

È il numero di bit per ogni secondo di suono, si misura in Kbps (kilo bit per secondo), di solito un segnale audio qualità CD ha un BitRate di 1411,2 Kbps; un file Mp3, invece, ha un BitRate che spazia da 112 Kbps a 320 Kbps.

```
End Sub

Private Sub MMControl1_NextCompleted(Errorcode
                                          As Long)
 impostaslider
End Sub
```

Infine nella *Form_Unload*, per sicurezza, conviene stoppare e chiudere la periferica aperta.

```
Private Sub Form_Unload(Cancel As Integer)
 MMControl1.Command = "stop"
 MMControl1.Command = "close"
End Sub
```

Fig. 3: **Ecco come si presenta il nostro lettore CD in esecuzione**

MP3, WMA E ALTRO

L'*MPEG* (*Moving Picture Export Group)* è ritenuto uno dei primi metodi intelligenti per la compressione e decompressione dei file audio (*CoDec audio*), da questo sistema è nato l'Mp3.
Attualmente ci sono valide alternative al formato Mp3, tra queste citiamo il formato *Windows Media Audio (Wma)* che a parità di qualità audio riesce a ridurre ulteriormente le dimensioni dei file; *OGG Vorbis* un ottimo codec open source; il codec della *Global Music Outlet* ed infine il formato *MPEG-4 AAC* che è supportato dalla Apple con i famosi *iPod* e *iTunes Music Store*.

CREAZIONE FILE MP3

Un modo semplice ed efficace per creare file compressi, da un *CdAudio*, è quello di utilizzare le funzionalità di conversione di *Windows Media Player (WMP)* o di *iTuner*, quest'ultimo è il Player lanciato dall'Apple.
Con *WMP* la creazione dei file *Mp3* o *Wma* s'imposta dalla scheda copia file da CD della finestra opzioni (selezionabile dalla voce di menu *Strumenti/Opzioni*). Con *ITuner*, invece, dopo aver definito le preferenze di conversione, si può convertire un file (non necessariamente contenuto in un CD) utilizzando un semplice menu *PopUp* attivabile con il tasto destro del Mouse.

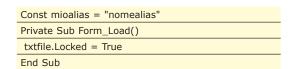
IL LETTORE MP3

In questo paragrafo presentiamo un lettore di file musicali realizzato con la sola funzione *mciSendString*. Il lettore permette di ricercare un file musicale, di eseguirlo, di stopparlo o di metterlo in pausa.

1 Nel progetto bisogna referenziate la libreria *Microsoft Common Dialog Control* che contiene il controllo *CommonDialog* da utilizzare per ricercare i file.
Sul form bisogna inserire un *CommonDialog*, un *textbox*, un controllo *Timer* e cinque pulsanti. Questi ultimi vanno nominati: *Play*, *Stop*, *Pause*, *Close* e *Cerca*. Il *Timer* è utilizzato per far scorrere, durante l'esecuzione, il nome del file contenuto nel *TextBox* (il codice da prevedere nel *Timer* lo trovate nel CD allegato alla rivista).

Fig. 4: **Ecco come si presenta la form per la realizzazione del lettore MP2**

2 Nella parte dichiarativa del form è prevista una costante, che contiene il nome dell'alias da attribuire al device. Nella form load, invece, è previsto il codice che blocca il Textbox.

```
Const mioalias = "nomealias"
Private Sub Form_Load()
 txtfile.Locked = True
End Sub
```

3 La ricerca di un file è fatta con la *cerca_Click*. La riproduzione del brano musicale, invece, si avvia con la *Play_Click*. In quest'ultima procedura, in particolare, si valuta il percorso corto del file, si individua il tipo di periferica da aprire (in base all'estensione del file), si apre la periferica con *OPEN*, s'imposta il formato dell'ora in millisecondi, si esegue il brano con *PLAY* e si gestiscono gli errori *MCI*. Gli errori sono catturati con la *GestErr* che richiama la *mciGetErrorString*.

```
Private Sub cerca_Click()
```

GLOSSARIO

MCISEND COMMAND
Un'altra funzione API che permette di controllare le periferiche audiovisive è la *mciSendCommand* che, a differenza della *mciSendString*, invia comandi basati su particolari strutture dati; per questo la gestione con la *mciSendCommand* è più macchinosa e non attinente con quella del controllo MCI. Nel progetto Visual Basic fornito con la rivista trovate un semplice esempio basato su questa funzione, si tratta di un form con due pulsanti che inviano i comandi per aprire e chiudere il supporto del lettore CD.

```
With CommonDialog1
 .DialogTitle = "Cerca un file sonoro"
 .Filter = "File multimediali |*.mpg;*.mpeg;" _
 & "*.avi;*.mpa;*.mpv;*.m1v;*.mp2;*.mp3" & _
 ";*.mpe;*.mpm;*.au;*.aif;*.wav;*.wma;*.midi;
                                          *.mid"
 'i file avi generano  un errore mci
 .ShowOpen
 If .filename <> "" Then
  txtfile.Text = .FileTitle
  Stop_Click
 End If
End With
End Sub

Private Sub Play_Click()
 Dim tipo As String
 Dim filename As String
 Dim lResult As Long
 Dim OpenMP3 As String
 If Me.txtfile = "" Then
  MsgBox "Specificare un file", vbCritical, "Attenzione"
  Exit Sub
 End If
 filename = pathnomecorto(
                      CommonDialog1.filename)
 Select Case UCase(Right(filename, 3))
  Case "MP3", "WMA":
   tipo = "MPEGVideo"
  Case "WAV":
   tipo = "Waveaudio"
  Case "MID":
   tipo = "Sequencer"
  Case Else
   MsgBox "Il file selezionato non è MP3, Wav,
                                    Wma o Mid"
   tipo = "Other"
 End Select

 lResult = mciSendString("OPEN " & filename
    & " TYPE " _& tipo & " ALIAS " & mioalias, 0, 0, 0)
 If lResult <> 0 And lResult <> 289 Then
  ' NB0 289 generato se prima era in Pausa
  OpenMP3 = GestErr(lResult)
  MsgBox OpenMP3
 End If
 mciSendString "SET " & mioalias & " TIME FORMAT
                                   MS", 0, 0, 0
 ' MS = millisecondi
 mciSendString "PLAY " & mioalias, 0, 0, 0
 Timer1.Interval = 500
End Sub

Private Function GestErr(lError As Long) As String
 Dim sBuffer As String
 sBuffer = String$(255, Chr(0))
 mciGetErrorString lError, sBuffer, Len(sBuffer)
 GestErr = Replace$(sBuffer, Chr(0), "")
```

```
End Function
```

Il codice da prevedere per i pulsanti *Pausa*, *Stop* e *Close* è il seguente.

```
Private Sub Pausa_Click()
 mciSendString "PAUSE " & mioalias, 0, 0, 0
 Timer1.Interval = 0
End Sub

Private Sub Stop_Click()
 Timer1.Interval = 0
 mciSendString "CLOSE " & mioalias, 0, 0, 0
End Sub

Private Sub close_Click()
 Timer1.Interval = 0
 mciSendString "CLOSE " & mioalias, 0, 0, 0
 txtfile = ""
End Sub
```

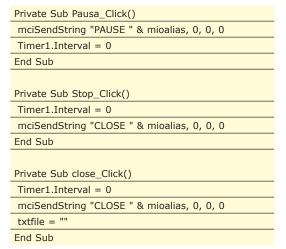



Fig. 4: **Il lettore mp3 in esecuzione**

Notare che la *Stop Click()* e *Close Click()* inviano lo stesso comando e che le due operazioni si differenziano grazie a *txtfile = ""*. Questa scelta è motivata dal fatto che per la *mciSendString* il comando *Stop* ha lo stesso effetto del comando *Pausa!* Inoltre notate il *Timer1.Interval =0* serve per bloccare il movimento della stringa. Infine per evitare problemi con la chiusura del form conviene prevedere il seguente codice.

```
Private Sub Form_Unload(Cancel As Integer)
 Stop_Click
End Sub
```

CONCLUSIONI

Il lettore CD potrebbe essere potenziato con l'introduzione di un file di testo (o di un database) che associa delle stringhe al numero di traccia. Il Player Mp3, invece, nel successivo appuntamento, sarà potenziato con la gestione della posizione di lettura e dei dati di esecuzione, con i tasti per controllare il volume degli altoparlanti e con l'aggiunta delle funzionalità che consentono di registrare dei file sonori. Nel successivo appuntamento, inoltre, mostreremo come riprodurre i file video, i DVD, come controllare una WebCam e come masterizzare.

Massimo Autiero

Reti Neurali PC che pensano

La replicazione dell'intelligenza umana da parte di una macchina è da sempre il sogno di scrittori di fantascienza e scienziati. In questo articolo "addestreremo" un computer a pensare...

"Sognerò?" è la domanda che HAL9000, il primo computer intelligente della storia cinematografica, pone al dottor Chandra nella consapevolezza di stare per essere spento. Citazione scontata, ma dovuta per un film che ha aperto il sogno dell'intelligenza artificiale. Macchine che pensano, in grado, nel bene e nel male, di prendere in autonomia le proprie decisioni. Macchine che elaborano dati non secondo una sequenza classica di *if/then/else* come siamo abituati a vedere in programmazione, ma che viceversa replicano il complesso meccanismo di neuroni, assoni, stimolazioni sinaptiche, che producono infine l'output: il pensiero umano.
In questo articolo tenteremo di replicare una rete neurale del tutto simile a quella presente nel cervello umano. Il nostro scopo finale sarà creare una rete tale che riesca a decidere da sola la risposta a una stimolazione di tipo booleano. Forniremo cioè alla nostra macchina in input due numeri, e "lei" sarà in grado di decidere sulla base di una relazione di *"OR"* logico qual è il valore booleano dell'operazione *OR*. Sembra una banalità, eppure la nostra sarà una piccola macchina pensante, una porta aperta verso l'affascinante oceano dell'intelligenza artificiale.

PARALLELISMO E PLASTICITÀ

Quando sono nate, le reti neurali si prefiggevano come obiettivo quello di "emulare" il comportamento del cervello umano. Ora sappiamo che tale obiettivo è ben lontano dall'essere raggiunto e probabilmente mai sarà raggiunto. Ad ogni modo il modello biologico rimane la fonte primaria d'ispirazione per la replicazione dell'intelligenza umana, per cui vale la pena comprenderne i principi base.
Tutti sappiamo che il nostro cervello è costituito da cellule chiamate neuroni. Queste unità computazionali di base formano tra loro delle interconnessioni che danno vita al più spinto ed efficiente "calcolatore parallelo" mai realizzato. Inoltre la grande quantità di cellule e di connessioni fanno si che il malfunzionamento di una di essa sia ininfluente ai fini del funzionamento del sistema complessivo.
Secondo un modello estremamente semplificato ogni neurone è costituito da:

1. Un corpo cellulare dove avvengono le reazioni bio-chimiche della cellula.
2. Dei dendriti, che sono dei corti filamenti, dai quali la cellula riceve gli stimoli elettrici da tutte le altre cellule a cui è collegata.
3. Un assone, per mezzo del quale la cellula propaga il suo segnale elettrico generato nel corpo.
4. Le sinapsi, che sono delle ramificazioni poste sulla parte terminale dell'assone, grazie alle quali può connettersi ai dendriti delle altre cellule.

Fig. 1: ***Uno schema sintetico della rappresentazione dei collegamenti neurali***

Cerchiamo ora di capire come funziona questo modello del neurone: molto semplicemente potremmo dire che ogni neurone riceve degli stimoli elettrici dalle altre cellule mediante i dendriti a cui, a loro volta, sono collegate le sinapsi di queste ultime. In

base agli stimoli ricevuti, il potenziale elettrico del corpo della cellula assume un certo valore; se tale valore supera una certa soglia, a sua volta la cellula propagherà un impulso elettrico lungo il proprio assone. In questo sistema un ruolo di fondamentale importanza lo rivestono le sinapsi: a seconda di quanto spesso due cellule scambiano segnali tra loro la relativa sinapsi può presentare un carattere eccitatorio od inibitorio. Infatti una sinapsi che viene "utilizzata" frequentemente tende a condurre il segnale molto meglio di quanto non lo faccia una utilizzata di rado. Da ciò si può comprendere come, benché l'uscita elettrica di un neurone (cioè l'assone) sia unica, grazie alle sinapsi ogni altra cellula riceverà un segnale d'intensità differente.
Come potete già cominciare a capire un'architettura che preveda migliaia di miliardi di unità di calcolo di questo genere presenta due caratteristiche fondamentali:

1. Dato un numero qualsiasi di neuroni e fissato il modo in cui essi sono interconnessi (ossia fissata la topologia della rete) il comportamento complessivo del sistema cambia radicalmente in funzione della conduttività delle sinapsi. Questo sarà anche più chiaro tra un po' quando sfrutteremo questa proprietà per addestrare la rete.
2. Ogni neurone lavora contemporaneamente a tutti gli altri ed allo stesso tempo dipende da tutti gli altri. In questo modo l'elevato parallelismo non comporta neanche problemi di comunicazione tra le unità di elaborazione.

DALLA BIOLOGIA AL SOFTWARE

Ora che abbiamo visto quali sono i principi base di una cellula neurale biologica tentiamo di imitare madre natura!

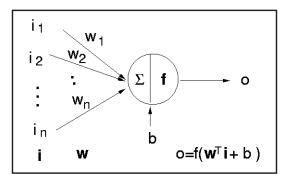



Fig. 2: *Uno schema di come il nostro neurone sarà definito al livello informatico*

Per quanto detto fino ad ora, abbiamo bisogno di tre elementi per "costruire" un neurone:

- Un insieme di pesi sinaptici (da ora in poi li chiameremo semplicemente pesi per brevità) che modellizzino la conducibilità delle sinapsi. Naturalmente ad ogni peso sinaptico è associato anche un segnale d'ingresso proveniente da un altro neurone.
- Un sommatore che si occupa di moltiplicare ogni segnale al suo relativo peso e sommi tutti i prodotti così ottenuti in modo tale da determinare il potenziale del neurone.
- Una funzione d'attivazione che riceva in ingresso il valore del potenziale e restituisca l'uscita del neurone stesso.

LA LIBRERIA SNARLI

SNARLI è una libreria Java sviluppata dal prof Simon D. Levy della Washington and Lee University ed ospitato su sourceforge all'indirizzo http://snarli.sourceforge.net/.
Tale libreria si basa, a sua volta, su un'altra libreria di algebra lineare (che magari avremo occasione di scoprire in uno dei prossimi articoli), che si chiama JLinAlg anch'essa ospitata su sourceforge all'indirizzo http://jlinalg.sourceforge.net/.
Per questo motivo avrete bisogno di entrambe le librerie per far funzionare correttamente le vostre reti. Il mio consiglio è quello di, se non siete interessati all'implementazione di tali librerie, di scaricare o prelevare da CD semplicemente i jar.

Questi tre aspetti sono ben rappresentati in **Figura 2**. Un po' d'attenzione va prestata al terzo punto dato che esso sarà di fondamentale importanza anche per il proseguo. La domanda che potrebbe sorgere è: che bisogno c'era di un'ulteriore funzione d'uscita? Senza tante considerazioni matematiche facciamo una considerazione puramente qualitativa. Supponiamo che un neurone cominci ad essere eccitato più di altri; quello che succederebbe è che i suoi pesi comincerebbero ad aumentare provocando a loro volta un aumento dell'uscita del neurone. Allo stesso modo, prima o poi, i neuroni eccitati ulteriormente da quest'ultimo andrebbero ad eccitare il neurone stesso instaurando così un ciclo infinito che porterebbe il sistema ad "esplodere" (o più propriamente a divergere): per evitare ciò si utilizza la funzione d'attivazione. La proprietà principale di questa funzione e quella di avere un'uscita limitata pur accettando qualsiasi range di valore in ingresso , per questo motivo tale tipo di funzioni è anche detto *squashing*.

DA PROGRAMMATORI AD INSEGNANTI!

Fino ad ora ci siamo concentrati sulle caratteristiche di un singolo neurone "isolato", che come avrete ben capito preso da solo non è poi così intelligente!
Ora quello che dobbiamo fare è cominciare a cambiare completamente il nostro approccio alla pro-

NOTA

COSA C'È DENTRO AL CERVELLO

Si pensi che nella sola corteccia di un cervello umano adulto si hanno all'incirca 10 miliardi di neuroni e che formano tra loro circa 60 mila miliardi di connessioni con le loro sinapsi. Questo elevatissimo parallelismo è sicuramente uno dei fattori che rende il nostro cervello l'organo più complesso mai studiato. Infatti, benché gli attuali transistor siano anche 6 - 9 ordini di grandezza più veloci dei neuroni, a tutt'oggi non si è riusciti a realizzare nulla che si avvicini alle capacità di un cervello anche di un animale.

grammazione. Indipendentemente da quale sia il vostro linguaggio preferito ed indipendente da quanto siano complessi i vostri programmi, su cosa si basa la cosiddetta programmazione dichiarativa o imperativa? Tutto si basa su quello che viene chiamato in linguaggio tecnico-informatico un algoritmo: che poi non è altro che una serie finita d'istruzioni per risolvere un problema dato. Quindi chiunque di voi abbia scritto un programma, per quanto piccolo esso fosse, ha scritto un algoritmo, pur non sapendolo! Il massimo della dinamicità pensabile per un algoritmo è una serie di *if/else* che permette di scegliere quale istruzione eseguire a seconda del verificarsi o meno di alcune condizioni. In altre parole siamo noi programmatori a prevedere il comportamento deterministico del programma stesso.

Da adesso faremo qualcosa di diverso: ci occuperemo solamente di costruire una struttura (la rete neurale) in grado di "imparare" da esempi che forniremo; non vi preoccupate tutto sarà più chiaro tra un po'. In effetti se pensiamo alla nostra esperienza comune possiamo dire che abbiamo due fondamentali modalità di apprendimento: una è quella che viene tipicamente applicata a scuola, ossia c'è un insegnante che ci trasmette delle informazioni e starà a noi assimilarle e farne buon uso. La seconda invece riguarda di più la vita quotidiana: se pensiamo ad esempio ad un'azione banalissima come quella del camminare ci accorgiamo che a nessuno di noi è stato detto di mettere un piede dopo l'altro ma con il tempo e dopo numerosissimi tentativi ed insuccessi abbiamo trovato il modo migliore per camminare. Anche in questo caso ingegneri e matematici hanno tratto ispirazione da queste semplici constatazioni per elaborare varie tecniche d'addestramento per le reti neurali. Come già accennato sopra possiamo dividere tutte le tecniche d'addestramento in due grandi categorie: *super-visionate* e *non super-visionate* che grosso modo corrispondono rispettivamente ai due esempi sopra riportati. In questo articolo ci occuperemo solo di reti *super-visionate* che forse sono quelle più facili sia da implementare che da capire.

COME ADDESTRARE LA RETE

In realtà fornire alla rete gli esempi è una delle parti più critiche per quanto riguarda le reti neurali. Ci sono due aspetti che dovete sempre tenere in considerazione: gli esempi devo essere il più generali possibile in modo tale che la rete possa poi comportarsi correttamente anche di fronte a casi mai visti finora. Il secondo aspetto da tenere in considerazione è che una rete tende a "specializzarsi", cioè impara molto meglio gli ultimi esempi rispetto ai primi; quindi è importante prestare attenzione all'ordine in cui essi vengono presentati.

LA FUNZIONE D'USCITA PER LO SQUASHING

Solitamente la funzione più utilizzata è la sigmoide. La sua espressione analitica è *f(x) = 1/(1+exp(-a*x))*. Come si vede questa funzione ha un range compreso tra 0 e 1. Un'altra proprietà utile è quella che tramite il parametro a possiamo modulare la "ripidità" della curva. Ci sono anche altre caratteristiche che rendono questa funzione molto utile, ma casomai ci torneremo in un prossimo articolo.

IMPARARE DAI PROPRI ERRORI

L'ultima cosa che ci rimane da fare prima di mettere mano ad un po' di codice è quella di riuscire a capire in che modo la nostra brava rete neurale possa venire addestrata. Esiste un'infinità di topologie di reti, quella che qui utilizzeremo si chiama *perceptrone* (o in inglese perceptron) ed è una delle prime comparse in letteratura. Pur avendo grossi limiti, ha dato il via ad altre reti più sofisticate, una delle quale avremo il piacere di presentare in un futuro articolo. Solitamente i vari i neuroni sono raggruppati in strati, o meglio *layers*. In un perceptrone ci sono esclusivamente due layers: uno di input l'atro di output.

Fig. 3: Una rappresentazione della rete di tipo "perceptrone"

Quello che dobbiamo fare prima di tutto è decidere quanto sia grande la rete, in questo caso di quanti ingressi e quante uscite abbiamo bisogno. Una volta stabilito ciò non dobbiamo far altro che fornire alla rete "alcuni" esempi; cioè è necessario fornire alla rete un serie d'ingressi e d'uscite desiderate. In questo modo possiamo definire una quantità che misuri l'errore per ciascuno dei neuroni d'uscita:

$$e[n] = d[n] - y[n]$$

dove n rappresenta il numero dell'esempio, y l'uscita effettiva della rete e d l'uscita desiderata. Ora possiamo sfruttare questa quantità per modificare il comportamento della rete in modo tale che "impari dai propri errori". In particolare, come abbiamo visto all'inizio, basterà modificare i pesi sinaptici in funzione dell'errore commesso in modo tale da far avvicinare il più possibile l'uscita effettiva a quella desiderata.

Più dettagliatamente definiamo un incremento (o decremento) di ogni peso come:

$$\Delta w_i[n] = \eta\, e[n]\, x_i[n]$$

dove x è un ingresso e l'indice i indica il numero dell'ingresso e del relativo peso, mentre η è detto *learning rate*. Questo parametro indica la percentuale di quanto i pesi debbano cambiare rispetto all'ingresso e all'errore, ma per ora tralasciamo questo aspetto; quello che ci interessa sapere è che all'esempio successivo i pesi verranno aggiornati in questo modo:

$$w_1[n+1] = w_i[n] + \Delta w_i[n]$$

L'ultimo problema è stabilire quando una rete ha terminato la fase d'apprendimento e può essere quindi pronta per essere utilizzata. Si dice che ogni volta che una rete abbia scorso tutti gli esempi e quindi aggiornato di volta in volta i pesi sia passata un epoca. Naturalmente occorre iterare più epoche prima di poter considerare la rete pronta. Determinare il numero di epoche non è un scienza esatta e dipende da molti fattori, solitamente si procede aumentandolo progressivamente fino a raggiungere le performance desiderate. Tenete conto anche che non sempre un numero elevato di epoche dia risultati migliori di un numero minore per motivi che vedremo (forse) la prossima puntata.
Ora siamo pronti per passare alla pratica!

UN PO' DI CODICE...

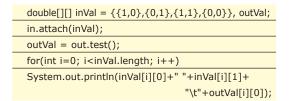
Ci appoggeremo alla libreria SNARLI, per ulteriori informazioni vi rimando al box *La libreria SNARLI*, e vedremo come creare una rete funzionante con un relativamente breve "script" Java. Quello che faremo è addestrare la nostra rete in modo tale che si comporti come una porta logica *OR*. Tale porta ha due ingressi ed una sola uscita ed è definita come segue:

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

La classe che utilizzeremo è *BPLayer* che rappresenta un layer di cui abbiamo prima parlato. Quindi possiamo procede così:

```
// creiamo un layer d'ingresso ed uno d'uscita
// specificando quanti neuroni essi debbano contenere
public static void main(String[] args) {
   BPLayer in = new BPLayer(2);
   BPLayer out = new BPLayer(1);
   //connettiamo l'uscita all'ingresso
   out.connect(in);
   //inizializziamo ogni peso sinaptico ad un numero
                                             casuale
   in.randomize(System.currentTimeMillis());
   out.randomize(System.currentTimeMillis());
   // prepariamo gli esempi come coppia
   // ingresso-uscita desiderata
   double[][] inPattern = {{0,0},{1,0},{0,1},{1,1}};
   double[][] outPattern = {{0},{1},{1},{1}};
   // forniamo gli esempi alla rete e diciamole di imparare
   in.attach(inPattern);
   out.attach(outPattern);
   out.batch(10,0.1,0);
```

il metodo *batch* riceve tre argomenti: il numero delle epoche, il *learning rate* ed il *momento*. Di quest'ultimo parametro non abbiamo discusso fino ad ora, sappiate comunque che ponendolo a zero non comprometterete mai il funzionamento della rete.
Facciamo lavorare la rete passandogli gli ingressi e stampando a schermo le risposte

```
double[][] inVal = {{1,0},{0,1},{1,1},{0,0}}, outVal;
in.attach(inVal);
outVal = out.test();
for(int i=0; i<inVal.length; i++)
System.out.println(inVal[i][0]+" "+inVal[i][1]+
                                  "\t"+outVal[i][0]);
```

Non male vero? Ora proviamo a sfruttare tutte le potenzialità di una rete neurale cambiando ad esempio il *training set* (gli esempi) ha soli tre casi e verificate se successivamente la rete risponderà correttamente, in fase di test, anche al quarto caso che non ha mai visto. In altre parole modifichiamo le righe della fase d'addestramento; una delle possibili scelte è:

```
double[][] inPattern = {{0,0},{1,0},{0,1}};
double[][] outPattern = {{0},{1},{1}};
```

Lasciando inalterato tutto il resto, come risponde la rete? Se avete settato in maniera opportuna i parametri dovrebbe rispondere correttamente anche all'ingresso 1 1.
Quello che vi consiglio è divertitevi a sottrarre alla rete esempi e vi accorgerete che un esempio non vale l'altro!

CONCLUSIONI

In questo articolo abbiamo cominciato ad addentrarci nel mondo delle reti neurali cercando di mettere in evidenza le caratteristiche che le distinguono dai paradigmi della programmazione classica ed abbiamo anche visto anche come grazie a *SNARLI* implementare una rete non sia poi così difficile.
Vi lascio con un quesito: al posto dell'*OR* provate ad addestrare la rete con uno *XOR* ossia

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

In questo caso la rete non si comporta proprio bene vero? Bé per chi non conoscesse già tutta la storia, se volete, potete confrontare le vostre idee in merito sul forum di ioProgrammo e comunque ci torneremo su nel prossimo articolo.

Andrea Galeazzi

IL CALCOLO PARALLELO NON È SEMPLICE

Uno delle più grandi sfide della teoria del calcolo parallelo è proprio l'efficienza nel coordinare le varie unità elaborative. In linea di principio si potrebbe pensare ad esempio che utilizzando un'architettura costituita da un altissimo numero di processori fornisca prestazioni elevatissime.
In realtà, oltre che a problemi energetici e di dissipazione de calore, esiste un problema di gestione del parallelismo stesso: è infatti necessario impiegare uno o più algoritmi che gestiscano tutti i problemi che possono insorgere nel calcolo parallelo, come ad esempio conflitti d'accesso alla memoria etc...
Così alla fine si arriva in un punto in cui, in termini computazionali, nell'esecuzione di tali algoritmi si spende più di quanto si guadagni con l'aggiunta di un altro processore.

Far funzionare bene la memoria

La gestione della memoria, è stato uno dei compiti più pesanti per gli sviluppatori. Con l'avvento dei meccanismi di gestione automatica, sarà il garbage collector a fare il lavoro sporco

Ogni applicazione del mondo reale, durante il suo ciclo di vita, dovrà utilizzare delle risorse, siano esse dei file, delle connessioni di rete, dei database o delle generiche locazioni di memoria. Nel paradigma di programmazione orientato agli oggetti, una generica risorsa è rappresentata da un oggetto di una data classe. Se abbiamo bisogno di connetterci ad un database SqlServer, utilizzeremo in .NET la classe SqlConnection, che rappresenta la connessione ad una sorgente dati. Per creare una connessione, è necessario creare un'istanza della classe, in C# ad esempio mediante la famigerata istruzione *new*. L'istruzione non fa altro che allocare una certa quantità di memoria, tanto grande da contenere l'istanza da creare e che rappresenterà la risorsa stessa. Prima di utilizzare la connessione, l'area di memoria deve essere inizializzata, in maniera che l'istanza trovi tutto ciò di cui ha bisogno, e di ciò si occupa il costruttore della classe. A questo punto possiamo utilizzare la risorsa allocata, cioè l'istanza della classe, invocando ad esempio i suoi metodi, e quando non c'è più bisogno di essa ripulirne la memoria. Ecco, di quest'ultima operazione non dobbiamo preoccuparci, sarà il Garbage Collector a stabilire quando la connessione non è più utilizzata e quindi a liberare la memoria da essa occupata. Semplice, ma non sempre è così. Per tipi semplici come *String* o *Int32,* non è necessario nessun trattamento particolare, ma quando si ha a che fare con risorse come file o connessioni di rete o genericamente risorse che sono wrapper di risorse non gestite, il discorso cambia, e bisognerà prestare particolari cure all'operazione di clean-up della memoria.

REQUISITI

Conoscenze richieste
Basi di programmazione .NET

Software
.NET Framework SDK

Impegno

Tempo di realizzazione

LA MEMORIA HEAP

Il *Common Language Runtime* alloca la memoria necessaria ad ogni oggetto da creare, in un'area di memoria detta *heap*. La memoria heap è gestita totalmente dal CLR, nel senso che quando un oggetto non serve più, esso libera automaticamente la memoria da esso occupata, senza necessità di intervento dello sviluppatore da codice. Il CLR mette in funzione il Garbage Collector, che determina quando l'oggetto ha terminato il suo ciclo di vita. Ma qual è il momento giusto per attivare la garbage collection? E come fa il garbage collector a sapere che nessun altro avrà bisogno di un oggetto in seguito? Il CLR assume che la memoria heap sia infinita.Ogni volta che viene utilizzata l'istruzione *new* per creare un'istanza, esso calcola la quantità di byte necessari ad ospitare l'oggetto nello heap e verifica che tale memoria sia disponibile. Se ciò non fosse vero verrebbe generata un'eccezione di tipo *OutOfMemoryException.* In caso contrario invece l'oggetto viene allocato nel punto indicato da un puntatore detto *NextObjPtr,* che indica sempre in quale posizione dovrà essere posizionato un nuovo oggetto da creare. L'istruzio-

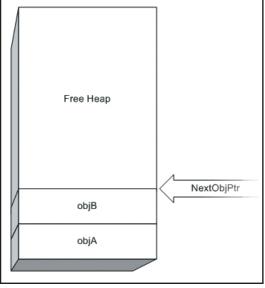



Fig. 1: La memoria heap con due oggetti

ne *new* restituirà l'indirizzo di memoria puntato da *NextObjPtr* ed il puntatore verrà spostato nel punto immediatamente successivo, all'oggetto appena creato nello heap. Proviamo a immaginare la memoria heap come un rettangolo, che viene allocato dal basso verso l'alto, e supponiamo che vi siano in memoria due oggetti *objA* e *objB*. La situazione apparirebbe come in **Figura 1**.

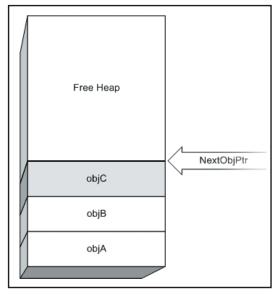



Fig. 2: ***Allocazione di un nuovo oggetto***

A questo punto immaginiamo che l'applicazione abbia creato un nuovo oggetto *objC*. Esso viene posizionato nel punto indicato da *NextObjPtr*, il quale verrà poi spostato subito dopo l'oggetto *objC*. Il garbage collector però non può entrare in azione solo quando il puntatore è oltre lo spazio disponibile, cioè non aspetta l'ultimo momento, ma per semplificare per ora pensiamo che sia così.

OGGETTI VIVI E NON

Quando si finisce di utilizzare un oggetto è bene liberarne la memoria, onde evitare cosiddetti *memory leaks,* a questo punto è necessario fare attenzione a non usare oggetti già *"disposed"*, incorrendo ad esempio in errori di puntatori nulli. Il garbage collector verifica che nella memoria heap non ci siano oggetti inutilizzati, in maniera da poterli rilasciare e quindi liberare la memoria che essi occupavano.
Per stabilire se un oggetto non è utilizzato da nessuno, ogni applicazione possiede un insieme di riferimenti roots (radici), che sono delle locazioni di memoria contenenti dei riferimenti a oggetti nello heap oppure degli oggetti *null*. Il *Common Language Runtime* distingue inoltre fra riferimenti root e riferimenti non root. Sono root ad esempio tutte le variabili statiche di un tipo riferimento, o ancora tutte le variabili locali di tipo riferimento o i parametri di metodi sullo stack, mentre riferimenti non-root sono ad esempio i campi di un'istanza.
L'esistenza di un riferimento root ad un oggetto è già sufficiente per affermare che esso è utilizzato dall'applicazione, mentre un oggetto senza riferimento root è un potenziale oggetto non più in uso.
Prima di andare avanti facciamo un esempio che chiarisca le idee, utilizzando un po' di codice, in maniera da togliere qualche dubbio su oggetti necessari e non necessari, e su come essi vengono giudicati tali. Consideriamo il seguente codice C#:

```
class Auto {
  public int velocita;
  public void Accelera(int v)
  {
    velocita=v;
  }
}
public class MyApp
{
  public static Auto stAuto==new Auto();
  static void Main()
  {
    //1. crea l'oggetto auto nello heap, alfa è un
                                      riferimento root
   Auto alfa=new Auto();
   for(int i=0;i<100;i++)
   {
     alfa.Accelera(i);
   }
   //2. Creo un nuovo oggetto mai referenziato
   Auto beta=new Auto();
   //3. alfa è ancora un riferimento root
   Console.WriteLine(alfa.velocita);
   //4. alfa è ancora nello scope, ma non è più
                                      referenziata
   Console.ReadLine();
  }
}
```

Il concetto di variabile *"viva"* o *"non viva"* non è basata sullo scope. Il CLR si basa su informazioni che il compilatore JIT gli fornisce, in maniera da sapere per ogni istruzione da eseguire, in anticipo, se un oggetto sarà ancora referenziato o meno.

- Nel **punto 1** l'oggetto *alfa* viene creato sullo heap, ed è un riferimento root, il compilatore JIT e il CLR infatti sanno che l'oggetto alfa verrà utilizzato nel ciclo *for* seguente.
- Nel **punto 2** viene creato un oggetto *beta,* ma esso non è un riferimento root, in quanto non sarà mai utilizzato, è il compilatore JIT ancora una volta che permette al CLR di conoscere

Utilizza questo spazio per le tue annotazioni

FINALIZATION QUEUE

Il meccanismo di garbage collection non effettua direttamente il rilascio della memoria degli oggetti non più utilizzati, ma inserisce tali oggetti in una coda di finalizzazione, da cui un thread secondario li pescherà e si occuperà della finalizzazione vera e propria.

questa informazione.

- Nel **punto 3**, *alfa* è ancora riferimento root, in quanto verrà utilizzato subito dopo nell'istruzione *WriteLine*.

- Nel **punto 4**, *alfa* è ancora nello scope, però non sarà più referenziata, quindi il Garbage Collector può anche liberarne la memoria.

Al di fuori del metodo *Main,* è stato creato un altro oggetto statico *stAuto,* in quanto statico esso potrebbe essere utilizzato in qualunque momento durante il ciclo di vita dell'applicazione, quindi è un riferimento root.

L'ALGORITMO DI GARBAGE COLLECTION

Ci sono dei problemi che i sviluppatori C/C++ o di un altro linguaggio senza meccanismi di gestione automatica della memoria si trovano a dover affrontare giornalmente, è che sono una delle principali fonti di bug e/o malfunzionamenti.
Nel framework .NET, il compilatore JIT e il CLR si occupano di mantenere la lista di tutti i riferimenti root dell'applicazione.
Per trovare gli oggetti non più utilizzati, il garbage collector utilizza un meccanismo a due fasi, detto di *Mark/Compact.*
Nella prima fase, il garbage collector di default assume che nessun oggetto fra quelli in memoria heap sia utilizzato, cioè parte dal presupposto che non esista nessun riferimento root verso oggetti nello heap.
Quindi inizia dal primo elemento della collezione di root e marca tutti gli oggetti dello heap che sono direttamente raggiungibili e, mediante una procedura ricorsiva, ogni oggetto che è riferito attraverso altri oggetti (**Figura 3**).

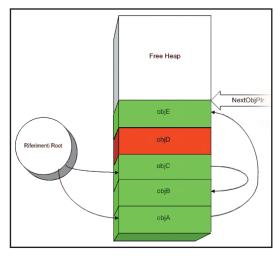



Fig. 3: Riferimenti root in memoria heap

E così per ogni riferimento root. Per incrementare le performance dell'algoritmo, quando si incontra un oggetto che è già stato inserito nel grafo, l'algoritmo continua con il root successivo, evitando di ripetere percorsi già visti e di entrare in qualche percorso circolare. Alla fine del procedimento, sarà stato creato un grafo contenente tutti gli oggetti in qualche maniera raggiungibile da un'application root, cioè tutti gli oggetti ancora utilizzati nell'applicazione.
Nella seconda fase, quella di *Compact*, il garbage collector può percorrere lo heap e liberare le locazioni di memoria di oggetti senza riferimenti. Se vengono trovati grossi spazi di memoria contigui, allora la memoria viene compattata e gli application root sono aggiornati con i nuovi indirizzi nello heap. I piccoli blocchi di memoria invece vengono saltati per incrementare la velocità dell'algoritmo.
Alla fine il puntatore *NextObjPtr* punterà al primo spazio libero immediatamente successivo all'ultimo oggetto in memoria, come in **Figura 4**, il blocco in rosso indica invece un oggetto non più utilizzato, cioè garbage da rimuovere.

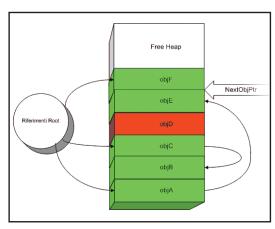



Fig. 4: La memoria heap dopo la creazione di un nuovo oggetto

È chiaro che con un meccanismo del genere, i malfunzionamenti accennati all'inizio del paragrafo possono essere dimenticati, in quanto non ci sarà necessità di ricordarsi di ripulire la memoria da oggetti non più utilizzati, e sarà impossibile utilizzare oggetti già rimossi, in quanto è il CLR che sa già quali oggetti devono ancora essere utilizzati.

GENERAZIONI DI OGGETTI

Il garbage collector di .NET utilizza dei concetti che si sono dimostrati utili e necessari per migliorare le performance di tutto il meccanismo. Esso non entra in azione quando tutta la memoria heap è esaurita , ed inoltre non tutti gli oggetti hanno la stesa importanza, magari ne esistono alcuni in

GLOSSARIO

DISPOSE PATTERN

Il processo di Garbage Collection è non deterministico. Nel senso che non possiamo mai sapere quando esso entrerà in funzione, anche se siamo noi a forzarlo, con i metodi della classe GC, ed inoltre non è prevedibile l'ordine con cui verranno "finalizzati" gli oggetti non più utilizzati. Se si ha necessità di un meccanismo deterministico che effettui il Dispose di un oggetto e delle risorse da esso utilizzate, soprattutto se si tratta di risorse unmanaged, dobbiamo implementare il cosiddetto pattern Dispose.

memoria da molto tempo, e dovranno ancora restarci per ancora più tempo. In definitiva, il garbage collector di .NET appartiene alla famiglia di garbage collector generazionali, che si basano su semplici ma dimostrabili assunzioni.

- Da meno tempo esiste un oggetto, tanto più breve sarà la sua vita.
- Da più tempo esiste un oggetto, tanto più lunga sarà la sua vita.
- Ripulire una piccola porzione di memoria richiede meno tempo che ripulirla tutta.

Quando la memoria heap viene inizializzata, essa sarà naturalmente vuota. I primi oggetti istanziati sullo heap faranno parte della prima generazione di oggetti, detta *generazione 0,* alla cui dimensione il CLR assegna anche una soglia massima, naturalmente minore della dimensione totale dello heap.
Supponiamo che la dimensione massima per la generazione 0 sia di 256KB, e quindi di creare un po' di istanze, come in **Figura 5**:

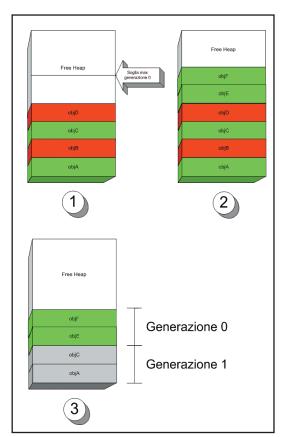



Fig. 5: ***Nuova generazione dopo una garbage collection***

Se la creazione di un nuovo oggetto farà superare la soglia della generazione 0, verrà azionato il garbage collector, che rimuoverà gli oggetti non più utilizzati (*objB* e *objD*) e farà incrementare la generazione degli oggetti restanti (*objA* e *objC*), in questo caso passeranno tutti alla *generazione 1*.
Se a questo punto vengono creati nuovi oggetti, essi faranno parte di una nuova generazione *0* (*objE* e *objF*). Quindi la *generazione 0* contiene sempre l'insieme delle istanze più giovani, quelle create più di recente, mentre, come detto, dopo ogni garbage collection la *generazione 0* è sempre vuota. Creando delle nuove istanze a questo punto si avrà una nuova *generazione 0*, e quando si raggiunge nuovamente la soglia di 256K della *generazione 0*, il garbage collector inizierà una nuova "passata".
Così come per la *0*, anche la *generazione 1* avrà una sua soglia massima, supponiamo di 1MB, ed ogni volta che si ha una garbage collection, deve essere deciso su quale generazione o generazioni di oggetti agire. Ciò dipende dalla memoria occupata da ognuna di esse. Se ad esempio la generazione 1 ha una soglia di 1MB, ed è occupata solo per 500Kb, la garbage collection avrà luogo solo sulla *generazione 0.* Questo per ottimizzare le performance del meccanismo agendo su una memoria più piccola.
Dopo questa seconda garbage collection, anche altri oggetti sopravvissuti, passeranno alla *generazione* 1, che quindi avrà una dimensione maggiore. Se il processo avesse interessato anche la *generazione 1*, cosa che avviene solo quando viene raggiunta la soglia di 1MB, ed in genere dopo molte garbage collection di *generazione 0,* i sopravvissuti della *1* sarebbero passati ad una generazione 2. La *generazione 2* quindi conterrà degli oggetti che sono stati esaminati come minimo 2 volte, cioè gli oggetti con l'età più alta.
Il processo di garbage collection attualmente implementato nel CLR supporta le generazioni 0,1 e 2 (vedi il prossimo paragrafo per ulteriori informazioni) ognuna con una dimensione massima, stabilita durante l'inizializzazione del Runtime, ed aggiornate ad ogni garbage collection in modo da ottimizzare le performance.

LA CLASSE SYSTEM.GC

Il garbage collector può essere controllato programmaticamente utilizzando la classe *System.GC* del .NET framework.
Come detto nel precedente paragrafo, il massimo numero di generazioni implementato attualmente è 3, che potete verificare leggendo la proprietà *MaxGeneration* della classe GC.

```
int maxGen=GC.MaxGeneration;
Console.WriteLine("max generation number:
                                {0}",maxGen);
```

- **C# AND .NET PLATFORM** ***Troelsen*** **(APress)**
- **APPLIED MICROSOFT .NET PROGRAMMING** ***Richter*** **(Microsoft Press)**
- **ESSENTIAL .NET, VOLUME I: THE COMMON LANGUAGE RUNTIME** ***Don Box, Chris Sells*** **(Addison Wesley)**

Attenzione al fatto che la proprietà restituisce il valore *2*, ad indicare il numero identificativo massimo, e cioè abbiamo tre generazioni possibili, la *generazione 0*, la *1* e la *2*.
La classe *GC* mette poi a disposizione il metodo *GetGeneration*, che restituisce la generazione di cui fa parte un oggetto.
Ad esempio definiamo una nostra classe *MyClass*, con il solo distruttore in maniera da avere un riscontro della sua effettiva finalizzazione:

```
class MyClass
{
   int id;
   public MyClass(int id)
   {
       this.id=id;
    }
    ~MyClass()
   {
       Console.WriteLine("oggetto distrutto");
   }
}
```

Se creiamo un'istanza della classe essa entrerà a far parte della generazione *0*, ed infatti se invochiamo il metodo *GetGeneration* ne avremo la conferma:

```
MyClass obj=new MyClass(0);
int objGen=GC.GetGeneration(obj);
Console.WriteLine("obj Generation: {0}",objGen);
```

Il meccanismo di garbage collection può essere forzato per mezzo del metodo *Collect*, di cui esiste una versione senza parametri per agire su ogni generazione, mentre passando un argomento intero, minore o uguale a *GC.MaxGeneration*, si agisce solo sulla generazione indicata.
Se dopo la costruzione dell'oggetto, eseguissimo ad esempio il codice seguente:

```
GC.Collect(0);
objGen=GC.GetGeneration(obj);
Console.WriteLine("obj Generation after Collect(0):
                              {0}",objGen);//passa alla gen 1
```

Otterremmo lo spostamento di *obj* alla *generazione 1*. Ed analogamente utilizzando ora il parametro *1*, l'istanza *obj* passerebbe in *generazione 2*, e non oltre naturalmente, anche con ulteriori invocazioni di *Collect*.
È bene non abusare del metodo *Collect*, in quanto sarà il CLR a stabilire il momento opportuno di farlo, sulla base di ciò che avviene nell'applicazione. In alcuni casi può però servire.
Ad esempio supponiamo di avere grosse moli di dati in memoria, sotto forma di oggetti naturalmente, e ad un certo punto effettuiamo un salvataggio su database. Potrebbe essere utile invocare il metodo *Collect* per forzare una garbage collection su tutte le generazioni, che certamente saranno piene di oggetti non più necessari perché salvati fisicamente.
La classe GC inoltre permette di conoscere la memoria approssimativa allocata. Il metodo statico *GetTotalMemory* restituisce il numero di byte allocati. Per esempio proviamo a farci restituire la memoria occupata prima e dopo un ciclo *for* che istanzia qualche migliaio di oggetti.

```
long freeMem=GC.GetTotalMemory(false);
Console.WriteLine("Total Memory: {0}",freeMem);
for(int i=1;i<100000;i++)
{
   MyClass o=new MyClass(i);
}
freeMem=GC.GetTotalMemory(false);
Console.WriteLine("Total Memory: {0}",freeMem);
```

provando ad eseguire il codice noterete un netto incremento della variabile *mem*.
Per completare la rapida carrellata sulla classe GC, mostriamo anche i metodi *SuppressFinalize* e *WaitForPendingFinalizers*. Quando degli oggetti posssono essere rimossi dalla memoria, essi vengono inseriti in una coda di finalizzazione.
Il garbage collector ad un certo punto invocherà il distruttore, cioè il metodo *Finalize*, di tali oggetti.
Il metodo *WaitForPendingFinalizers* sospende il thread in esecuzione fino a quando la coda di finalizzazione non è stata svuotata. Il metodo *SuppressFinalize* invece permette di rimuovere un metodo dalla coda di finalizzazione.

CONCLUSIONI

Normalmente non c'è alcun motivo di utilizzare manualmente il meccanismo di *Garbage Collector*, sarà sempre .NET a fare il lavoro sporco al posto nostro. Nei casi in cui sia opportuno ricorrere ad un'ottimizzazione dell'uso della memoria, la classe GC è l'unico ponte che ci viene offerto per entrare in contatto con il sistema. Senza contare che l'esistenza di questa classe ci consente di sviluppare applicazioni che possono produrre un'output contenenten il monitoring del controllo della memoria.
Si tratta di una delle funzioni più delicate del sistema, pertanto va utilizzata con cautela e tuttavia la conoscenza di come avvengono i meccanismi di gestione della memoria in .NET può condurci alla creazione di applicazioni estremamente ottimizzate.

Antonio Pelleriti

Potete contattare l'autore per suggerimenti, critiche o chiarimenti all'indirizzo e-mail antonio.pelleriti@ioprogrammo.it. **oppure sul sito** www.dotnetarchitects.it

Castor il gemello minore di Hibernate

Impareremo a realizzare applicazioni Java che fanno largo uso di database. Affideremo però tutta la gestione di SQL e delle relazioni a un tool OpenSource per poterci dedicare esclusivamente agli oggetti

In questo articolo realizzeremo in Java una piccola rubrica di indirizzi. Al di la dell'utilità indubbia di questo genere di applicazione, sarà importante notare come verranno risolti i problemi di *"persistenza dei dati"*. In particolare, utilizzeremo *Castor*, una libreria OpenSource che implementa un framework per mappare i dati presenti in un database relazionale in corrispondenti classi e oggetti conformi alla programmazione OOP. Il meccanismo di funzionamento è del tutto simile a quello implementato da Hibernate.

DALL'IDEA AL PROGETTO

Nella nostra rubrica l'elemento principale è il *Soggetto* - il nominativo da registrare; Il *soggetto* sicuramente sarà dotato di due attributi: *nome* e *cognome*. Opzionalmente potranno essere presenti indirizzi email, indirizzi fisici e numeri di telefono. Per mantenere la più alta flessibilità possibile (considerando anche la persistenza sarà eseguita in automatico tramite Castor e dunque non richiederà particolari sforzi di programmazione), si è deciso che questi elementi saranno implementati in entità diverse. Il modello dei dati della rubrica è dunque composto dalle seguenti classi

Soggetto;

- *Email;*
- *Indirizzo;*
- *NumeroTelefono.*

Il Soggetto contiene i seguenti elementi:

- *id*
- *nome;*
- *cognome;*
- *telefoni;*
- *email;*
- *indirizzi.*

Email, Indirizzo e *NumeroTelefono* derivano da una superclasse comune astratta, *DatoSoggetto,* che contiene gli attributi:

- *id;*
- *soggetto;*
- *oggetto.*

In queste classi il campo *id* è la chiave univoca, di tipo *long*, che identifica univocamente ciascun soggetto, email, indirizzo o numero di telefono. In *DatoSoggetto,* l'attributo soggetto è un riferimento al soggetto a cui appartiene quell'elemento. Si immagini l'indirizzo email *mario@rossi.it:* l'oggetto che rappresenta questa email è legato direttamente al nominativo *"Mario Rossi"*. Questo legame permette di sapere che, ad esempio, questo indirizzo non appartiene al nominativo *"Alberto Bianchi"*. Il campo *oggetto* contiene invece una descrizione che permette di distinguere diversi account di email, indirizzi o numeri di telefono. Ad esempio, *Mario Rossi* potrebbe avere anche la mail *mariorossi@soletech.it;* in questo caso l'*oggetto* varrà *"lavoro"*, mentre nell'*oggetto* che contiene *mario@rossi.it* oggetto sarà *"casa"*. Si può pensare

COME INIZIARE

È necessario avere ovviamente installato il J2SE1.5, gli altri file notevoli sono castor-0.9.6.tgz da scompattare e porre in una directory del classpath di Java. All'interno del classpath sono necessarie il file *xerces.jar* che fa parte del pacchetto *Xerces-J-bin.1.4.4.tar.gz* - http://xml.apache.org/xerces-j/ - e la libreria per il *common logging commons-logging.jar* che fa parte del pacchetto *commons-logging-1.0.4.tar.gz* – http://jakarta.apache.org/site/downloads/downloads_commons.html - Potete scaricare tutto dal web oppure ovviamente utilizzare il software contenuto nel cd allegato alla rivista.

all'*oggetto* come una dicitura che fa distinguere le diverse email, indirizzi o numeri di telefono.
In particolare, queste tre entità dispongono di una serie di attributi. La classe *Email* contiene solo il campo *email,* che contiene il nome dell'account; la classe *Indirizzo* contiene invece:

- *via;*
- *numero;*
- *cap;*
- *città;*

la classe *NumeroTelefono* contiene invece solo:

- *prefisso;*
- *numero;*

L'IMPLEMENTAZIONE

La classe *Soggetto* è implementata come una normale classe Java, che però deve rispettare, al fine di ottenere la persistenza automatica con Castor, una serie di convenzioni. In particolare:

- deve essere presente la chiave univoca, infatti è implementato l'attributo *id* di tipo *long;*
- deve essere presente un costruttore di default (quello senza parametri);
- ogni attributo che deve essere reso persistente deve avere *setter/getter* (in realtà Castor permette anche un accesso diretto, ma si è scelto di seguire comunque la strada che prevede l'uso di *getter* e *setter,* in quanto più allineata alle convenzioni del mondo Java;

La classe è così implementata:

```
package it.edmaster.rubrica.model;
public class Soggetto {
  long id;
  String nome;
  String cognome;
  NumeroTelefono[] telefoni;
  Email[] email;
  Indirizzo[] indirizzi;
  public Soggetto() {
  }
  public Soggetto( String nome, String cognome ) {
     this.nome = nome;
     this.cognome = cognome;
  }
  public String getIntestazione() {
     return nome + " " + cognome;
  }
  public String toString() {
     return getIntestazione();
  }
  //... getter e setter omessi
}
```

si noti della presenza del metodo *getIntestazione(),* che ritorna una concatenazione del nome e del cognome del soggetto, che verrà utilizzata come intestazione della scheda del nominativo; è presente anche l'implementazione del metodo *toString(),* che non fa altro che ritornare l'intestazione. Quando è possibile, è utile implementare questo metodo, definito nella classe *Object,* in modo che, stampando un oggetto, si ottenga una descrizione dello stesso. Ad esempio:

```
Soggetto s = new Soggetto("Mario Rossi");
System.out.println(s);
```

stampa:

Mario Rossi

Se non fosse stato implementato *toString(),* si avrebbe ottenuto il comportamento standard, presente nella classe *Object,* che presenta una cosa del tipo:

Soggetto@B382

Il codice completo (meno *getter* e *setter*) della classe *DatoSoggetto* è il seguente:

```
package it.edmaster.rubrica.model;
public abstract class DatoSoggetto {
  Soggetto soggetto;
  String oggetto;
  long id;
  /** richiesto dalle sottoclassi */
  public DatoSoggetto() {}
  public DatoSoggetto(Soggetto soggetto, String
                                            oggetto) {
     this.soggetto = soggetto;
     this.oggetto = oggetto;
  }
  public abstract String[] getContenuto();
//getter e setter omessi
}
```

si noti il costruttore vuoto, richiesto dalle sottoclassi, ed il metodo astratto *getContenuto(),* che viene implementato dalle sottoclassi in funzione del tipo di dato che contengono. Questo metodo ritorna un array di stringhe che rappresentano il contenuto; nel caso delle email, viene ritornato semplicemente un array di un elemento con, ad esempio, *mario@rossi.it.* Nel caso degli indirizzi la cosa è più complessa, visto che il contenuto si di-

DOCUMENTAZIONE DI CASTOR

La documentazione della struttura dei file XML di Castor è presente all'indirizzo http://www.castor.org/jdo-mapping.html, dove è presente il reference di tutti i tag che è possibile posizionare nei file di configurazione.

spone su più righe. Il fatto di avere questo metodo a livello di superclasse permette di strutturare le classi che implementano l'interfaccia utente in modo da trattare in generale con *DatoSoggetto*, e non in particolare con *Email, Indirizzo* o *NumeroTelefono*.
Questo è uno degli aspetti potenti della OOP, che consente di risparmiare codice, si vedrà più avanti come.

CONFIGURARE CASTOR

La cosa "magica" nei toolkit di persistenza come Castor, Hibernate o Apache ObjectRelational Bridge è che sono in grado di eseguire la creazione, cancellazione ed aggiornamento delle entità su base dati semplicemente analizzando le classi dell'applicazione con la reflection ed utilizzando una manciata di informazioni di configurazioni presenti in file XML di supporto.
In particolare, Castor utilizza un file che per default si chiama *database.xml*, e che nel caso di questo progetto è il seguente:

```
<!DOCTYPE databases PUBLIC "-//EXOLAB/Castor
                JDO Configuration DTD Version 1.0//EN"
                "http://castor.exolab.org/jdo-conf.dtd">
<database name="rubrica" engine="mysql" >
    <driver url="jdbc:mysql://localhost/rubrica"
                class-name="org.gjt.mm.mysql.Driver">
        <param name="user" value="root" />
        <param name="password" value="mysql" />
    </driver>
        <mapping href="support/mapping.xml" />
</database>
```

come si nota osservando il contenuto, Castor è configurato per memorizzare i dati in un database MySQL. Si notano infatti gli URL relativi al server, il nome del database ("rubrica") e le informazioni di autenticazione necessarie ad accedere al server.
Al termine di *database.xml* è presente poi un riferimento ad un ulteriore file: *mapping.xml*. Questo contiene la configurazione sulla persistenza di ciascuna classe. Ad esempio, *Soggetto* è definito come segue:

```
<class name="it.edmaster.rubrica.model.Soggetto"
            identity="id" key-generator="IDENTITY">
    <description>Soggetto</description>
    <map-to table="SOGGETTI" />
    <field name="id" type="long">
      <sql name="ID" type="integer" />
    </field>
    <field name="nome" type="string">
      <sql name="NOME" type="char" />
    </field>
    <field name="cognome" type="string">
      <sql name="COGNOME" type="char" />
    </field>
```

come si nota, l'elemento principale *<class>* si aspetta un attributo *"name"* che specifica la classe, completa di package, a cui si sta facendo riferimento. Molto importante sono i due attributi seguenti:

- **identity -** Indica qual è il campo Java (non la colonna del database) che contiene la chiave dell'identità della classe. L'identità è il campo che individua univocamente una istanza di questa classe e corrisponde sul database alla chiave primaria;
- **key-generator -** Questo attributo è indispensabile per definire come Castor crea una nuova chiave primaria per questa classe. In questo caso è stato utilizzato l'attributo *IDENTITY*, che sostanzialmente utilizza la strategia nativa del database, ma ci sono anche altri algoritmi dove è Castor il principale responsabile per la creazione della nuova chiave. Utilizzando MySQL è possibile utilizzare i campi interi con attributo *auto_increment*, che hanno la caratteristica di generare sempre un nuovo numero ad ogni inserimento nella tabella.

All'interno di *<class>* è presente l'elemento *<map-to>* che indica la tabella su cui memorizzare i dati. Sono presenti anche uno o più elementi *<field>*, che definiscono, insieme agli elementi figli *<sql>*, quali campi della classe devono essere resi persistenti e come. Specificano anche il tipo nativo (presente in *<field>*) e quello SQL (presente in *<sql>*). L'attributo *"name"* specifica il nome del campo (in *<field>*) ed il relativo nome di colonna (in *<sql>*).
Oltre ai campi basilari visti sopra, il mapping della classe Soggetto prevede anche degli oggetti figli, che sono in relazione *1:n*. Per configurare questa relazione è sufficiente specificare l'attributo *collection*, indicandone il tipo e, all'interno di *<field>*, specificare il tag *<sql>* indicando la chiave esterna nell'attributo *"many-key"*. In questo caso, il campo delle tabelle per gli elementi figli contengono sempre un campo *ID_SOGGETTO* che contiene l'ID del soggetto a cui appartengono. Questa è una informazione indispensabile per porre in relazione la tabella *SOGGETTO* con le altre del database:

```
<field name="email" type="it.edmaster
            .rubrica.model.Email" collection="array">
        <sql many-key="ID_SOGGETTO"/>
</field>
<field name="telefoni" type="it.edmaster.rubrica
```

NOTA

CASTOR, XML ED LDAP

Castor non è solo persistenza, ma anche esportazione ed importazione dei dati in XML ed accesso ad LDAP. Quest'ultima è una tecnologia per accedere a server specifici che memorizzano informazioni in modo gerarchico.

```
        .model.NumeroTelefono" collection="array">
        <sql many-key="ID_SOGGETTO"/>
    </field>
    <field name="indirizzi" type="it.edmaster.rubrica
                .model.Indirizzo" collection="array">
        <sql many-key="ID_SOGGETTO"/>
    </field>
</class>
```

in questo caso le collezioni sono *array*, ma Castor supporta anche *Vector, Hashtable, Collection, ArrayList, Set* e *Map*. La configurazione della relazione deve essere completata nella definizione della classe collegata a *Soggetto;* ad esempio, si analizzi la configurazione della classe *Indirizzo:*

```
<class name="it.edmaster.rubrica.model.Indirizzo"extends=
        "it.edmaster.rubrica.model.DatoSoggetto" depends=
        "it.edmaster.rubrica.model.Soggetto"identity="id">
    <description>indirizzo</description>
    <map-to table="INDIRIZZO" />
    <field name="via" type="string">
      <sql name="VIA" type="char" />
    </field>
    <field name="numero" type="string">
      <sql name="NUMERO" type="char" />
    </field>
    <field name="cap" type="string">
      <sql name="CAP" type="char" />
    </field>
    <field name="citta" type="string">
      <sql name="CITTA" type="char" />
    </field>
    <field name="soggetto" type="it.edmaster
                .rubrica.model.Soggetto">
      <sql name="ID_SOGGETTO" />
    </field>
</class>
```

si noti il campo *"soggetto":* viene indicato che è di tipo *it.edmaster.rubrica.model.Soggetto,* cosa che chiude, insieme all'attributo *"depends"* dell'elemento *<class>*, il giro della relazione *1:n* con la tabella *SOGGETTI.*
A questo punto è necessario far capire a Castor che le classi *Email, Indirizzo* e *NumeroTelefono* sono in realtà delle sottoclassi di *DatoSoggetto,* la cui definizione è la seguente:

```
<class name="it.edmaster.rubrica.model
                .DatoSoggetto" identity="id"
                key-generator="IDENTITY">
    <map-to table="DATI_SOGGETTO" />
    <field name="id" type="long">
      <sql name="ID" type="integer" />
    </field>
    <field name="oggetto" type="string">
      <sql name="OGGETTO" type="char" />
    </field>
</class>
```

come si nota, anche in questo caso viene utilizzata l'identità del database; nella colonna *OGGETTO* viene memorizzato l'oggetto, mentre ci si potrà chiedere perché manca *ID_SOGGETTO.* D'altra parte, se l'ID di *DATI_SOGGETTO* permette di andare sulle tabelle delle sottoclassi, ad esempio *EMAIL*, per ottenere la riga relativa, perché *ID_SOGGETTO* non è in *DATI_SOGGETTO* ma in ciascuna tabella delle sottoclassi? Questa è una limitazione di Castor, che costruisce le chiavi SQL in modo da puntare direttamente alle tabelle figlie; costruire una query che accedesse anche a *DATI_SOGGETTO* sarebbe stato molto più complesso e forse non portabile su tutti i database server.
L'ultimo passo è creare un file .SQL con i campi che servono all'applicazione. Per brevità ne riportiamo un estratto. Il db completo è comunque disponibile nel codice allegato alla rivista

```
CREATE TABLE dati_soggetto (ID int(11) NOT NULL
    auto_increment, OGGETTO varchar(40) default NULL,
                PRIMARY KEY  (ID)) TYPE=MyISAM;
CREATE TABLE email (ID int(11) NOT NULL
        auto_increment, EMAIL varchar(40) NOT NULL
    default '', ID_SOGGETTO int(11) NOT NULL default
                '0', PRIMARY KEY  (ID)) TYPE=MyISAM;
```

Esiste qualche metodo per creare automaticamente il file .SQL partendo dal file di mapping, tuttavia al momento utilizzeremo una configurazione manuale.

PERSISTENZA DEI DATI

Per introdurre i dati nel database sarebbe necessaria una complessa interfaccia utente, dunque in questa fase vengono creati dei soggetti di prova, istanziando direttamente le classi di dominio. Ad esempio, il codice seguente crea il soggetto *Marco Mari*, dotato di due indirizzi email, un numero di telefono e dell'indirizzo di casa e del lavoro:

```
Soggetto s2 = new Soggetto("Marco", "Mari");
emails = new Email[2];
emails[0] = new Email(s2, "personale",
                "mmari@email.it");
emails[1] = new Email(s2, "lavoro",
                "mmari@genertech.com");
s2.setEmail( emails );
telefoni = new NumeroTelefono[1];
telefoni[0] = new NumeroTelefono( s2, "cell.1",
                "347", "1236475" );
s2.setTelefoni( telefoni );
indirizzi = new Indirizzo[2];
```

LAYOUT MANAGER

Una delle caratteristiche più potenti – ma complesse – di Java è che questi non posiziona i componenti dell'interfaccia utente in modo assoluto, ma utilizzando degli oggetti – i layout manager – che contengono una logica di dimensionamento e posizionamento degli oggetti. Questo garantisce la portabilità delle interfacce sui diversi sistemi operativi, ma richiede anche più codice.

```
indirizzi[0] = new Indirizzo(s2, "casa", "Via Larici",
                    "7", "Corsico (MI)", "20100");
indirizzi[1] = new Indirizzo(s2, "lavoro", "Via De
                    Amicis", "200", "Milano", "20131");
s2.setIndirizzi( indirizzi );
```

la classe *CreaSoggetti* crea tre soggetti, e li persiste utilizzando le API di Castor. In particolare, è necessario utilizzare una istanza dell'oggetto *Database,* avviando la transazione con *begin()*, creando l'oggetto con *create()* e concludendo con *commit()* e *close()*. Castor delimita ogni operazione sulla base dati racchiudendole tra i metodi *begin()* e *commit():*

```
db.begin();
Soggetto[] soggetti = new Soggetto[] { s1, s2, s3 };
for (int i=0; i<soggetti.length; i++) {
        db.create( soggetti[i] );}
db.commit();
db.close();
```

l'oggetto *Database* necessario ad operare con Castor viene ottenuto nel programma utilizzando una classe di supporto, *CastorUtils:*

```
package it.edmaster.rubrica.util;
import org.exolab.castor.jdo.Database;
import org.exolab.castor.jdo
                    .DatabaseNotFoundException;
import org.exolab.castor.jdo.*;
import org.exolab.castor.mapping.MappingException;
public class CastorUtils {
  JDO jdo;
  private CastorUtils() {}
  public static CastorUtils getInstance() {
    return new CastorUtils();
  }
  public Database createSession() throws
                    DatabaseNotFoundException,
                    PersistenceException {
    if( jdo == null ) {
        jdo = new JDO();
    jdo.setDatabaseName("rubrica");
      jdo.setConfiguration("support/database.xml");
        jdo.setClassLoader(
                    getClass().getClassLoader() );
    }
    Database database = jdo.getDatabase();
    return database;
  }
}
```

CastorUtils è un *singleton,* infatti implementa il metodo *getInstance()* che ritorna un oggetto di questa classe, che contiene anche il metodo *createSession()*. Questo metodo crea una istanza di *Database* utilizzando un oggetto *JDO* che viene inizializzato impostando il nome del database, il percorso del file di configurazione ed un *class loader,* che sarà quello utilizzato per accedere alle classi rese persistenti.

INTERFACCIA UTENTE

Arrivati a questo punto è abbastanza semplice creare un'interfaccia utente adeguata. I dati possono essere recuperati dal database attraverso castor con una sintassi del tipo:

```
soggetto.getTelefoni()
soggetto.getEmail().length + 1
```

Una query al database può essere fatta attraverso *OQL (Object Query Language),* molto simile a SQL, con una sintassi del tipo

```
public ElencoSoggetti() throws
    DatabaseNotFoundException, PersistenceException {
  OQLQuery query;
  Database db = CastorUtils.getInstance()
                    .createSession();
  db.begin();
  query = db.getOQLQuery("SELECT s FROM it
                    .edmaster.rubrica.model.Soggetto s");
  QueryResults results = query.execute();
```

Il risultato di una query è un oggetto *QueryResults,* che viene convertito in una lista in questo modo:

```
  elenco = new ArrayList( results.size() );
  while( results.hasMore() ) {
    elenco.add( results.next() );
  }
```

alla fine di tutto, nonostante questa sia una operazione di lettura e non di scrittura, è necessario eseguire una *commit()* ed un *close():*

```
        db.commit();
        db.close();
}
```

CONCLUSIONI

Castor è un ottimo tool di persistenza dei dati. Non ha la classe e le capacità di Hibernate ma è senza dubbio più semplice da configurare e utilizzare e i definitiva rappresenta un'ottima soluzione per chi necessita di flessibilità senza voler installare un tool dalla complessità elevata quale è Hibernate. A fronte di un minimo di difficoltà iniziale si hanno prestazioni elevate.

Massimiliano Bigatti

IL PROGETTO CASTOR

È disponibile all'indirizzo http://www.castor.org/, da cui si può accedere alla documentazione di progetto, al Wiki, forum ed al download del codice e degli esempi.

Codice C++ per Windows e Linux

L'obiettivo di C++ fin dalla sua nascita è stato, tra gli altri, quello di essere multipiattaforma. Oggi grazie a wxWidgets è davvero possibile scrivere codice complesso che funzioni su tutti i sistemi

Il punto è il seguente: programmare applicazioni facilmente portabili da un sistema operativo a un altro. È per questo che è nato C++, è questo che ha fatto la fortuna di Java, ed è ancora questo che si tenta di fare con Mono. Scrivere applicazioni multipiattaforma significa abbracciare l'intero mercato e non una nicchia con la conseguenza di allargare il proprio bacino d'utenza in modo esponenziale.
In questo articolo abbiamo scelto di realizzare un programma C++, assolvendo alle esigenze di multipiattaforma tramite l'uso un *MiddleWare* che si pone come ponte fra il sistema operativo e le funzionalità di alto livello, come la gestione della grafica, il file system, i servizi di rete. Tale MiddleWare risponde al nome di *WxWidgets*. In parole molto povere tutte le nostre chiamate e le nostre routine di programmazione faranno adesso riferimento a funzioni esposte da *WxWidgets*. Sarà questa libreria ad interfacciarsi con il sistema e a trasformare tutto in modo tale che le nostre applicazioni funzionino sia su Windows sia su GTK+, Motif e Mac. Ovviamente bisogna ricompilare l'applicazione per il sistema corretto, e bisogna disporre della libreria installata sul sistema. Niente paura, è tutto OpenSource. In questo articolo utilizzeremo come strumento di sviluppo *DevCpp* in una versione un po' particolare che è già integrata con *WxWidgets*, perciò davvero non ci sono costi da sostenere per sviluppare utilizzando questa tecnica, altro punto a suo favore.

COME INIZIARE

E necessario avere installato le *wxWidgets* relative al proprio ambiente di programmazione. Trovate ovviamente tutto sul cd di ioProgrammo, oppure sul sito http://www.wxwidgets.org/. Se siete in ambiente Linux è utile installare tutto da uno dei repository disponibili per la vostra distribuzione. Ancora, in ambiente Windows abbiamo utilizzato il compilatore *DevCPP*, in ambiente Linux potere utilizzare ovviamente gcc. Infine sempre in ambiente Windows abbiamo preferito l'ottimo *wxDevCpp* che consente di sviluppare applicazioni *wxWidgets* in modo del tutto visuale, in ambiente Linux ci siamo fatti aiutare da *Anjuta* che dispone di comodi wizard per la realizzazione di applicazioni *wxWidgets*.

PRIMI PASSI IN WXWIDGETS

La struttura delle classi e delle applicazioni di *wxWidgets* richiama molto quella di MFC.
Risulta, dunque, molto semplice effettuare il porting di applicazioni scritte con il framework di Visual c++ in wxWidgets. L'intera applicazione viene gestita da un oggetto di tipo *wxApp*. La classe *wxApp* è astratta e definisce due metodi *OnInit e OnExit*, di cui effettueremo l'override in fase di implementazione.
Il contenuto del *main.h* della nostra prima applicazione assomiglierà a qualcosa del genere:

```
#include <wx/wxprec.h>
#include <wx/wx.h>
class myApp :public wxApp
{
public:
        bool OnInit();
        int OnExit();
};
```

Ovviamente la classe sarà implementata nel file *main.cpp*. L'implementazione della classe astratta *wxApp* inizia con la riga

```
IMPLEMENT_APP( myApp )
```

e segue con la definizione dei metodi

```
bool myApp::OnInit()
{
    wxFrame *myWin = new  wxFrame(NULL,-1,
                                   "Hello wxWorld");
    SetTopWindow(myWin);
    myWin->Show(TRUE);
    return TRUE;
}
int myApp::OnExit()
{
    return 0;
}
```

La vista principale viene gestita dalla classe *wxFrame*. Questa classe fornisce i metodi per la gestione di una finestra, la cui dimensione e posizione può essere gestita dall'utente. Inoltre la finestra sarà dotata di bordi e di una barra del titolo. Compilando, il risultato è quello di ottenere una finestra di dialogo vuota che sulla caption bar mostra la scritta *"Hello World"*. Tanto per prendere confidenza con la logica di *wxWidgets,* aggiungiamo alla finestra una status bar e definiamo un testo di stato. Il codice è il seguente:

```
#include "main.h"
IMPLEMENT_APP(myApp)
bool myApp ::OnInit()
{
    wxFrame *myWin = new wxFrame(NULL, -1,"Hello
                                          wxWorld");
    myWin->CreateStatusBar();
    myWin->SetStatusText("Hello World");
    myWin->Show(TRUE);
    SetTopWindow(myWin);
    return TRUE;
}
```

Anche in questo caso il codice è abbastanza esplicativo. La classe *wxFrame* ci mette a disposizione i metodi *CreateStatusBar* e *SetStatus* che rispettivamente creano la barra di stato e vi aggiungono un testo.

QUALCOSA DI PIÙ COMPLESSO

La nostra applicazione è ancora scarna. Andiamo ad aggiungere un po' di elementi.
Per prima cosa associamo al frame un *Sizer* poi gli altri controlli. I *Sizer* sono particolari oggetti che si occupano della disposizione dinamica dei controlli e somigliano molto ai Layer di Java.

```
wxBoxSizer* box =
new wxBoxSizer(wxHORIZONTAL);
box->Add( new wxTextCtrl(myWin, ID_TEXTBOX ,
                 "Testo", wxPoint(0,0),wxSize(100,20) ),
0,wxALIGN_CENTER_VERTICAL | wxALL, 5);

box->Add( new wxButton(myWin, ID_BOTTONE,
             "Bottone", wxPoint(0,0), wxSize(80,20)),
           0,wxALIGN_CENTER_VERTICAL | wxALL,5);

myWin->SetSizer(box);
```

Abbiamo definito un sizer orizzontale che occupa l'intero spazio della finestra.
All'interno del sizer abbiamo aggiunto due controlli uno di tipo TextBox, che abbiamo riempito con un testo di prova: *"testo"*, abbiamo poi aggiunto un bottone, la cui caption, con grande fantasia, sarà *"Bottone"*.
Il risultato del codice appena scritto è un frame contenente un textbox e di seguito, sulla stessa riga, un bottone distanziati tra loro di 5 pixel.
Ovviamente se provate a compilare il codice in questione, non funzionerà. Manca la definizione degli ID delle risorse. L'abbiamo inserita in un file *myFrame.h* che contiene, per ora:

```
#define ID_TEXTBOX 1001
#define ID_BOTTONE 1002
```

definiremo qui gli ID che utilizzeremo nel corso dell'applicazione. Questi ID devono essere univoci e avranno un ruolo fondamentale nella gestione degli eventi.

QUALCHE CONTROLLO IN PIÙ

Alla nostra applicazione mancano ancora un una combobox e per finire una listbox. Il codice che le implementa è il seguente:

```
wxString scelte [] = { "Ciao" , "Questa", "è" , "una",
                                        "ComboBox" };
  wxString scelteLista [] = { "Ciao" , "Questa", "è" ,
                                   "una", "ListBox" };
  box->Add( new wxComboBox(myWin , ID_COMBO,
      "",wxDefaultPosition,  wxDefaultSize, 5, scelte,
       wxCB_DROPDOWN ),1, wxEXPAND| wxALL,5);
  box->Add( new wxListBox(myWin, ID_LISTA,
     wxDefaultPosition, wxDefaultSize, 5, scelteLista,
            wxLB_SINGLE ),1, wxEXPAND| wxALL,5);
```

Anche qui il tutto è abbastanza intuibile. Si deve porre attenzione alla dichiarazione dei due array che contengono le stringhe che dovranno riempire la combobox e la listbox. I due array vengo-

Utilizza questo spazio per le tue annotazioni

NOTE

COMPILARE IN AMBIENTE LINUX

In ambiente linux è ovviamente possibile compilare a linea di comando. Per quanto riguarda i nostri esempi è possibile compilare con la seguente stringa. Tenendo conto che i caratteri prima e dpo *wx-config* sono apici rovesciati ottenibili tramite la combinazione di tasti *ALTGR*+?

```
gcc `wx-config --cxxflags
 –libs` main.cpp myFrame
                .cpp -o main.o
```

no passati come parametro rispettivamente per i controlli *wxComboBox* e *wxListBox*.
Ancora una volta poniamo l'accento sull'ID che identifica il controllo e che viene definito con un numero nel file di inclusione *myFrame.h*, sarà indispensabile ricorrere a questo id nella gestione degli eventi. Perciò tenete sempre come regola base che ogni controllo deve essere identificato da un ID univoco.

MENU E BARRE DEGLI STRUMENTI

Risulta molto semplice anche aggiungere i menu e le toolbar. Vediamo subito un esempio sui menu. Andiamo nel corpo del costruttore della classe *myFrame* vista in precedenza ed aggiungiamo il menu.

```
wxMenuBar * menu = new wxMenuBar();
  wxMenu * esempio = new wxMenu();
  esempio->Append(ID_MENU , "&Apri\tCtrl-A" ,
                              "Apri un File" );
  esempio->AppendSeparator();
  esempio->Append( ID_ESCI , "&Esci\tEsc",
                         "Fine del Programma" );
  menu->Append( esempio , "&File" );
  myWin->SetMenuBar( menu );
```

La prima cosa da fare è creare una *Menubar*, all'interno della quale creeremo i singoli menu. Ogni elemento dei menu è caratterizzato da un ID, un titolo ed una descrizione. l'ID viene utilizzato per identificare l'oggetto, il titolo è il testo visualizzato mentre il testo della descrizione appare sulla statusbar quando la voce del menu in questione è selezionata. Il titolo della voce del menu può contenere una serie di caratteri speciali. Il primo che incontriamo è la &. Con questo carattere possiamo definire la lettera selezionabile con la combinazione *Maiuscolo-Tasto*.
Poi troviamo la sequenza *\tTasto* la quale ci permette di definire un tasto di scelta rapida.
Ormai dovreste avere appreso perfettamente come fare per aggiungere un toolbar. Sempre nel corpo del costruttore andiamo ad inserire il seguente codice:

LEGGERE IL CONTENUTO DEI CONTROLLI A RUNTIME

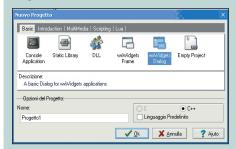
Per leggere il contenuto dei controlli a runtime possiamo utilizzare gli id a loro associati.

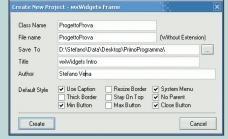
```
wxWindow *win = wxWindow ::
      FindWindowById( ID_TEXTBOX );
wxTextCtrl *text =
      wxDynamicCast(win, wxTextCtrl);
if ( text ){
  wxMessageBox(
         "Il Testo è il Seguente : \n" +
                    text->GetLabel() );
}
```

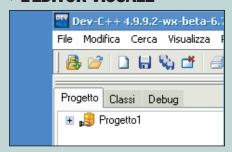
Ogni controllo è ereditato dalla classe *wxWindow*. La funzione *wxWindow::FindWindowById* restituisce un puntatore di tipo *wxWindow* tramite l'utilissima macro per il cast dinamico andiamo ad effettuare la conversione in *wxTextCtrl*. Se il cast è andato a buon fine leggiamo il testo del textbox e lo inseriamo in un finestra di messaggio. Gli ID sono le targhe dei controlli è importante, quindi, associarne uno, in modo univoco, ad ogni oggetto della GUI.

IL MIO PRIMO PROGETTO CON WXDEVCPP

Per il nostro articolo in ambiente Windows abbiamo utilizzato l'ottimo wxDevCpp che ci consente di usare i controlli esposti da wxWidgets in modo completamente visuale, esattamente come avremmo fatto da Visual Studio, ma ad un costo decisamente inferiore

> SI INIZIA

1 Una volta eseguito wxDevcpp selezioniamo dal menu File la voce Nuovo->Progetto. Dalla sezione basic scegliamo di creare un progetto dal template "wxWidgets Dialogs" inseriamo un nome per il nostro lavoro e diamo l'ok! Siamo già a buon punto.

> IMPOSTAZIONI INIZIALI

2 Il prossimo passaggio è definire il nome dei file e delle classi che compongono il progetto di base. Una volta completate le procedure di configurazione siamo pronti per iniziare a lavorare. DevCpp creerà per noi uno scheletro di applicazione da cui iniziare

> L'EDITOR VISUALE

3 *wxDevcpp* salva le informazioni sul layout grafico in file di estensione wxform.
È sufficiente andare nel visualizzatore dei file del progetto e selezionare i file con tale estensione per accedere all'editor di dialoghi e lavorare in modo visuale.

```
wxToolBar * tool =  myWin->CreateToolBar();
    wxBitmap *bitmap = new wxBitmap() ;
    bitmap->LoadFile("bottone.bmp",
                                  wxBITMAP_TYPE_BMP );
    tool->AddTool(ID_ESCI, "Esci", *bitmap, "Fine del
                                          Programma");
    tool->Realize();
```

Per prima cosa andiamo a creare la toolbar poi creiamo la bitmap da usare come immagine per il bottone della toolbar. Una volta pronta la bitmap andiamo a creare il pulsante della barra.
La ToolBar si comporta come la MenuBar, ovvero gestisce lo stesso tipo di eventi.
Per questo motivo è possibile condividere gli ID delle voci del menu con la barra delle utilità.
Con questo stratagemma è possibile eseguire una data operazione sia dal menu che dalla toolbar gestendo gli eventi una sola volta.

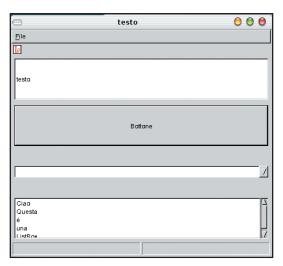
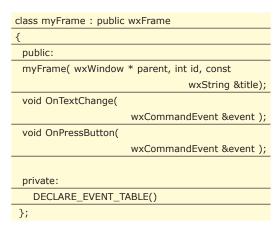



***Fig. 1:** L'applicazione di esempio in esecuzione su un'installazione di Ubuntu Linux*

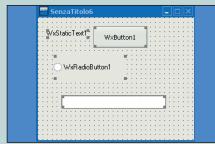
GLI EVENTI

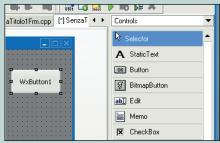
La gestione degli eventi in *wxWidgets* è allo stesso tempo semplice e potente.
Il primo passo sarà estendere le classi base di *wxWidgets* ed aggiungervi le funzionalità necessarie ai nostri scopi. Proprio per far fronte alle nostre esigenze estendiamo la classe *wxFrame* per poter ricevere le notifiche degli eventi.

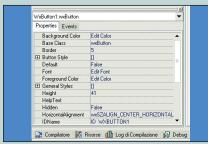
```
class myFrame : public wxFrame
{
  public:
  myFrame( wxWindow * parent, int id, const
                              wxString &title);
  void OnTextChange(
                      wxCommandEvent &event );
  void OnPressButton(
                      wxCommandEvent &event );

  private:
    DECLARE_EVENT_TABLE()
};
```

Ciò che abbiamo appena fatto è stato inserire le due funzioni *OnTextChange* e *OnPressButton* le quali accettano come parametro una variabile di tipo *wxCommandEvent*. Queste funzioni si comportano come dei callback.
La prima funzione viene richiamata quando il testo in un box di testo cambia, la seconda viene invocata nel momento in cui l'utente preme un pulsante.
La macro *DECLARE_EVENT_TABLE()* racchiude l'abilitazione alla ricezione degli eventi.
Occupiamoci ora dell'implementazione di *myFrame*. All'interno del file di implementazione è necessario inserire la seguente lista di macro

CROSS-PLATFORM

Con il termine cross-platform vengono indicati i progetti che possono essere ricompilati sotto piattaforme differenti. Scrivere codice completamente portabile è molto complicato poiché ogni sistema operativo mette a disposizione strumenti per lo sviluppo differenti (Win32, gtk, kde, ...). Anche la struttura e la dimensione dei tipi di base può cambiare tra compilatori diversi . wxWidgets ci viene incontro mettendoci a disposizione tutti gli oggetti e gli strumenti considerati "critici" evitandoci fastidiosi errori.

> DISEGNARE L'INTERFACCIA

4 In questa modalità è possibile inserire nel dialogo tutti gli oggetti messi a disposizione dal framework. Nell'esempio in figura abbiamo aggiuntio un bottone, un radiobutton una label e un textbox, in pratica tutti i controlli base a disposizione.

> COSA POSSIAMO FARE?

5 Tutti i controlli (e non solo) inseribili nei frame sono elencati nella barra a destra dell' IDE. Ovviamente stiamo parlando di un editor visuale che ci consente di disegnare semplicemente posizionando gli elementi sul frame che caratterizza la finestra.

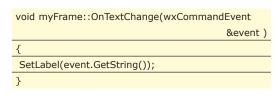
> L'EDITOR DI PROPRIETÀ

6 È possibile modificare i parametri di ogni controllo selezionato attraverso l'utile editor di proprietà in stile Visual Studio. lavorando sui diversi parametri otterremo comportamenti anche sensibilmente differenti e senza avere messo mano al codice.

```
////Event Table Start
BEGIN_EVENT_TABLE(myFrame,wxFrame)
EVT_TEXT( ID_TEXTBOX , myFrame::OnTextChange)
EVT_BUTTON(ID_BOTTONE, myFrame::OnPressButton)
EVT_MENU(ID_MENU, myFrame::OnApri)
END_EVENT_TABLE()
////Event Table End
```

La macro *BEGIN_EVENT_TABLE* prepara una lista di eventi e l'associa ad una classe. Nella suddetta macro è necessario specificare, come secondo parametro, la classe padre. Dopo è il turno della dichiarazione degli eventi *EVT_ TEXT,EVT_BUTTON*. Il primo evento viene generato ogniqualvolta il testo della textbox con ID *ID_TEXTBOX* cambia. Il secondo è generato dalla pressione del pulsante con ID *ID_ BOTTONE*. Gli eventi in questa fase vengono associati alle funzioni di callback della classe che stiamo implementando. Nel corpo delle funzioni di callback possiamo andare a gestire il carattere delle nostre applicazioni. Andiamo a gestire ad esempio l'evento *OnTextChange*.

```
void myFrame::OnTextChange(wxCommandEvent
                                    &event )
{
 SetLabel(event.GetString());
}
```

Il progetto *wxWidgets* è disponibile all'indirizzo http://www.wxwidgets.org/

Tutte le informazioni su *wxDevCpp* sono reperibili al seguente indirizzo. http://wxdsgn.sourceforge.net/

L'esempio è banale, leggiamo la stringa associata alla variabile *event* e la utilizziamo per cambiare il testo nella barra del titolo della nostra applicazione. Per un elenco completo dei metodi della classe *wxCommandEvent* vi rimando alla guida in linea di *wxWidgets*, tuttavia *wxCommandEvent* non è l'unica tipologia di evento gestibile.

- È possibile catturare la pressione dei tasti attraverso gli eventi di tipo *wxKeyEvent*.
- Le voci dei menu generano eventi di tipo *wxNewEvent*.
- Il mouse da origine a eventi di tipo *wxMouseEvent*.
- Possiamo anche "ridisegnare", secondo il nostro gusto, gli sfondi degli oggetti , quando lo permettono, attraverso l'intercettamento degli eventi di tipo *wxPaintEvent*.

Ecco un esempio: per prima cosa aggiungiamo l'evento alla *"Event Table"*

```
EVT_PAINT(myFrame::OnPaint)
```

poi lo implementiamo

```
void myFrame::OnPaint(wxPaintEvent& event)
{
  wxPaintDC dc(this);
  dc.DrawRectangle(10,10,100,50);
  dc.DrawText("Io Programmo",, 25  );
}
```

La dichiarazione di una variabile di tipo *wxPaintDC* all'inizio di un evento *OnPaint* è obbligatoria. Essa contiene il device context nel quale si può disegnare e scrivere.

DIALOGHI DI INPUT

Tra le centinaia di classi messe a disposizione dal framework di cui ci stiamo occupando c'è ne sono una serie dedicate all'input.
Le più interessanti ci sono il selettore di file, il *DirBrowser*, e il dialogo per la scelta dei colori. Andiamo subito a vedere il *FileSelector*

```
wxString filename =  wxFileSelector("Scegli il file");
if ( !filename.empty() )
{
  wxMessageBox( "Il file è :" + filename );
}
```

semplice no? Da notare l'utilità dell'operatore + per la concatenazione delle stringhe.
Stessa sintassi per il selettore di directory. L'unica cosa che cambia è il nome della funzione. In questo caso dobbiamo utilizzare *wxDirSelector*.
Per quanto riguarda la selezione di un colore bisogna aggiungere qualche elemento. In *wxWidgets* i colori sono codificati con il metodo RGB e gestiti tramite la classe *wxColour*.

```
wxColour colore ( 27, 127, 255 );
colore = wxGetColourFromUser(this,colore);
```

CONCLUSIONI

In questo articolo non siamo scesi nel dettaglio dei parametri relativi ai singoli metodi, questo tipo di informazione è facilmente reperibile nella documentazione allegata alle *wxWidgets*, abbiamo preferito invece dare uno sguardo d'insieme alla logica di funzionamento della libreria, fornendovi un tutorial che vi offre tutte le informazioni necessarie per costruire da soli un'intera applicazione.
Le wxWidgets sono uno strumento piuttosto potente e consentono di costruire codice realmente multipiattaforma, particolarità assolutamente non trascurabile.

Stefano Vena

Una form per ogni finestra

Le Windows Form rappresentano la struttura di base per lo sviluppo di applicazioni Microsoft Windows. Impariamo come utilizzarle, di quali proprietà godono e gestirle senza problemi

All'interno di un progetto di tipo Applicazione per Windows (abbiamo già visto come in VB .NET sia possibile creare altri tipi di progetti, ad esempio le Applicazioni Web), le Windows form (finestre Windows) rappresentano l'oggetto di base per l'interazione con l'utente. La corretta progettazione del posizionamento dei controlli sul form stabilisce il successo di un'interfaccia utente.
Le form sono degli oggetti e come tale espongono le proprietà che ne definiscono l'aspetto, ed i metodi e gli eventi che definiscono l'interazione con l'utente.

LA PRIMA FORM

Siamo ormai degli esperti nel creare nuovi progetti di tipo Applicazione per Windows, ed ogni volta abbiamo osservato come VB .Net crea automaticamente una form per iniziare a disegnare l'interfaccia utente. In concreto VB .NET crea una soluzione dal nome assegnatogli, costituita da un progetto dallo stesso nome, composto a sua volta da una finestra vuota denominata *Form1.VB*. In automatico, inoltre, mostrerà la finestra *Progettazione Windows Form* in cui sarà visualizzata la finestra *Form1*. Nella finestra *Progettazione Windows Form*, è possibile progettare rapidamente e visivamente il disegno di una form trascinando semplicemente i controlli, selezionati nella *Casella degli strumenti*, nella form. L'ambiente di progettazione di Visual Basic .Net è diventato, in realtà, un potente generatore di codice. Quando si disegna un controllo o si imposta una sua proprietà (dalla finestra delle *proprietà*), vengono generate, in automatico, le istruzioni che consentono di disegnare ed assegnare i valori delle proprietà. Tale codice viene racchiuso in una regione *"collapsed"* in modo che non sia possibile modificarlo per sbaglio. Per verificare quanto abbiamo appena detto, possiamo aprire l'editor di codice facendo doppio clic sulla form (oppure nella finestra *Esplora soluzioni*, selezionando *Form1* e scegliendo la voce *Visualizza codice*) e cliccare sulla casellina con all'interno il segno di più *(+)* a sinistra della casella con scritto *Codice generato da Progettazione Windows Form*. Con queste operazioni verrà visualizzato il codice che VB .Net 2003 genera in automatico. Per i più curiosi, consiglio di dare uno sguardo al codice, ma ricordiamoci di non modificarlo.

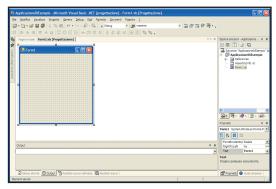



Fig. 1: **La finestra di progettazione**

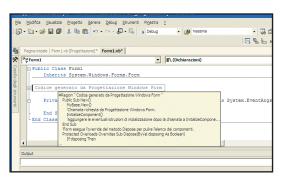



Fig. 2: **La regione "collapsed" del codice**

LE PROPRIETÀ

La proprietà tipica di una form è la proprietà *FormBorderStyle* che ci consente di definire l'aspetto del bordo della finestra. I valori che può assumere sono:

- **None** - non viene visualizzato nessun bordo e

NOTA

La proprietà *Name* consente di definire il nome con cui verrà fatto riferimento alla form nel progetto. Ogni volta che viene creata una nuova form, VB per impostazione predefinita, assegna i nomi *Form1*, *Form2* e così via. È consigliabile impostare, tramite la proprietà *Name*, un nome più significativo.

nessuna barra del titolo e la finestra non può essere spostata, ridimensionata o ridotta ad icona. Nelle applicazioni reali questo tipo di form non è d'uso comune.

- **FixedSingle** - usato per creare una form a dimensione fissa, viene visualizzata una casella con il menu di controllo, i pulsanti di riduci ad icona, ingrandisci e chiudi e la barra del titolo. La finestra non può essere ridimensionata (trascinando il bordo o l'angolo) ma può essere spostata, massimizzata oppure ridotta ad icona.
- **Fixed3D** - è uguale allo stile *FixedSingle* eccetto che la finestra appare con un effetto tridimensionale intorno ai bordi. La finestra non può essere ridimensionata ma può essere spostata, massimizzata oppure ridotta ad icona.
- **Sizable** - è il valore di default. Vengono visualizzati tutti i pulsanti: di riduzione ad icona d'ingrandimento e di chiusura, nonché la barra del titolo e la casella di controllo.
- **FixedDialog** - usato anch'esso per creare una form a dimensione fissa, vengono visualizzati i pulsanti di riduci ad icona, ingrandisci e chiudi e la barra del titolo. La finestra non può essere ridimensionata ma può essere spostata, massimizzata oppure ridotta ad icona.
- **FixedToolWindow** - visualizza la barra del titolo ed il pulsante di chiusura. La finestra può essere spostata ma non può essere ridimensionata o ridotta ad icona.

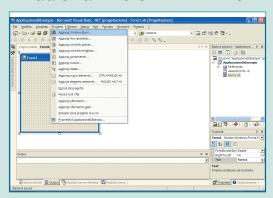
NOTA

Microsoft Windows gestisce il concetto di form secondaria. Se tra due finestre esiste una relazione di tipo padre-figlio, la form secondaria apparirà sempre davanti a quella principale, indipendentemente da quale delle due sia la form attiva. Per dichiarare che la finestra chiamata appartiene alla finestra chiamante occorre invocare il metodo *AddOwnedForm*

Scegliendo la voce di menu *progetto/ Aggiungi Form ereditato,* si aggiunge una Windows Form che eredita da una classe *Form* precedentemente creata. Vedremo nei prossimi articoli il concetto di *ereditarietà di classi,* ci basti sapere che la creazione di nuove finestre mediante l'eredità da form di base è un modo semplice per duplicare ciò che è stato creato senza ripetere ogni volta le stesse operazioni.

CREAZIONE DI UNA NUOVA FORM

Nella maggior parte delle applicazioni non sarà sufficiente la sola finestra creata in automatico da VB Net, ma ci occorreranno certamente altre form. Impariamo come aggiungere ulteriori form al progetto

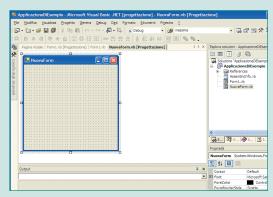
> AGGIUNGIAMO UN ELEMENTO

1 Per creare una nuova form in fase di progettazione, selezionare dal menu Progetto la voce Aggiungi Windows Form. Sarà visualizzata la finestra di dialogo Aggiungi nuovo elemento.

> SCEGLIAMO IL TEMPLATE

2 Nella finestra di dialogo Aggiungi nuovo elemento, fra i vari template esistenti selezioniamo il modello Windows Form (nella parte destra della maschera).

> DIAMOGLI UN NOME

3 Nel campo Nome dobbiamo digitare il nome della finestra (non è necessario digitare l'estensione di file .VB poiché viene aggiunta da Visual Basic). Potremo chiamarla NuovaForm.

> SIAMO PRONTI

4 Clicchiamo sul pulsante Apri. In questo modo si aprirà la finestra di progettazione con la nuova form pronta per essere usata. Non ci resta che aggiungere il codice che deve gestirla.

- **SizableToolWindow -** visualizza la barra del titolo ed il pulsante di chiusura. La finestra può essere spostata e ridimensionata.

Dopo aver definito il bordo della finestra possiamo definire l'aspetto della barra del titolo (sempre nei limiti imposti dalla proprietà *FormBorderStyle*). Utilizzando la proprietà *Text* possiamo indicare la stringa che dovrà essere visualizzata nella barra del titolo della form. Modificando la proprietà *ControlBox* possiamo decidere se far comparire i pulsanti di controllo della form. Si può inoltre scegliere quali pulsanti di controllo devono essere disabilitati, con la proprietà *MaximizeBox* per il pulsante d'ingrandimento e *MinimizeBox* per il pulsante di riduzione ad icona. Se ambedue le proprietà sono settate a *False* i pulsanti non vengono neppure mostrati a video. Tra le altre proprietà possiamo utilizzare:

- **Icon** - La proprietà Icon permette di selezionare l'icona che dovrà essere visualizzata nell'angolo superiore sinistro quando viene mostrata la form, oppure quando viene ridotta ad icona in fase di esecuzione. Per inserire un'icona, dobbiamo visualizzare la finestra delle proprietà e cliccare con il mouse sul pulsante (con i tre puntini) che appare quando si seleziona la proprietà *Icon*, così facendo verrà visualizzata la finestra di dialogo *Apri* in cui selezionare il file corrispondente all'icona desiderata.
- **WindowState** - La proprietà *WindowState* ci permette di definire lo stato di visualizzazione della form (ingrandita, ridotta ad icona o con dimensioni normali) e può assumere tre valori.
- **Normalper** - visualizzare una form con le dimensioni normali (è il valore di default).
- **Minimized -** per visualizzare una form ridotta ad icona.
- **Maximized** - per visualizzare una form a tutto schermo.
- **StartPosition** - La proprietà *StartPosition* permette di specificare la posizione che dovrà occupare la form sullo schermo. Può assumere quattro valori.
- **Manual -** non viene specificata nessuna posizione per la finestra.
- **CenterParent** - la finestra viene visualizzata al centro rispetto alla relativa form padre.
- **CenterScreen** - la finestra viene visualizzata al centro dello schermo.
- **WindowsDefaultLocation** - la finestra viene visualizzata nella posizione predefinita di Windows, con le dimensioni indicate.
- **WindowsDefaultBounds** - la form viene visualizzata nella posizione predefinita di Windows, con le dimensioni determinate dai limiti delle impostazioni predefinite di Windows.
- **AutoScroll** - Se una form contiene un numero di controlli tali, da non rientrare nei limiti imposti dalla risoluzione video, è possibile adottare due soluzioni: utilizzare strutture a schede oppure form scorrevoli.

Le strutture a schede rappresentano, da molto tempo, la modalità standard per raccogliere un gran numero di controlli in un'area ridotta dello schermo.
Le form scorrevoli possono essere invece utilizzate in molte situazioni, in modo particolare oggi che gli utenti sono abituati a scorrere su internet le lunghe pagine HTML. In VB.NET è molto semplice implementare una finestra scorrevole, è infatti sufficiente impostare a *True* la proprietà *AutoScroll.* Ogni volta che l'utente ridimensiona la finestra, in modo da rendere uno dei controlli parzialmente invisibili, verrà visualizzata una barra di scorrimento orizzontale o verticale (o entrambe) a seconda del caso.
Possiamo prendere confidenza con le proprietà appena descritte, modificandole nella finestra della proprietà, per avere subito un riscontro visivo sull'effetto che hanno sulla form.

GLI EVENTI

Analizziamo ora gli eventi di una form introducendoli nell'ordine in cui vengono generati lungo il ciclo di vita.

New - L'evento *New* è il primo evento generato nella vita di ogni form e determina lo stato di *Creato,* ma non caricato. L'evento viene generato nel momento in cui viene utilizzata la parola chiave *New* nel codice prima dell'invocazione del metodo *Show* (come vedremo nel proseguo dell'articolo). Il codice di quest'evento normalmente non è visibile poiché è racchiuso nella regione collassata *"Codice generato da Progettazione Windows Form".* In questa parte di codice si possono inserire le istruzioni necessarie all'inizializzazione delle variabili della form.

FORM DI AVVIO

Quando si disegna un'applicazione con diverse finestre si pone il problema di definire la finestra di avvio del programma. Per default la finestra di avvio è la prima che si è creata, per modificare l'ordine di esecuzione si devono seguire i seguenti passi:

1 Nella finestra Esplora soluzioni fare clic con il pulsante destro del mouse sul progetto e scegliere Proprietà.

2 Viene visualizzata la finestra di dialogo Pagine delle Proprietà NomeProgetto (dove NomeProgetto è il nome del progetto in uso) alla proprietà Generale

3 Cliccare sull'elenco a discesa Oggetto di avvio per visualizzare l'elenco di tutte le finestre del progetto

4 Selezionare dall'elenco la form che si vuole venga visualizzata per primo e cliccare sul tasto OK

La form di avvio possiede una peculiarità, se viene chiusa, vengono automaticamente chiuse anche tutte le altre finestre e l'applicazione termina.

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

```
Public Sub New()
    MyBase.New()
    'Chiamata richiesta da Progettazione Windows Form.
    InitializeComponent()
    'Aggiungere le eventuali istruzioni di
                      inizializzazione dopo la chiamata a
    'InitializeComponent()
End Sub
```

È sconsigliato scrivere codice in questo evento (a meno di non esserne costretti), ed usare piuttosto l'evento successivo, l'evento *Load*.

Load - Dopo l'evento *New* viene generato l'evento *Load*, in questa fase i controlli della form sono stati tutti creati e caricati anche se la finestra non viene ancora visualizzata. Si possono modificare e leggere le proprietà dei controlli, ma si devono evitare azioni che non possono essere eseguite su controlli invisibili. In genere questo evento contiene il codice in cui si inizializzano controlli e variabili.

```
Private Sub Form1_Load(ByVal sender As
          System.Object, ByVal e As System.EventArgs)
                                    Handles MyBase.Load
    TextBox1.Text = ""
End Sub
```

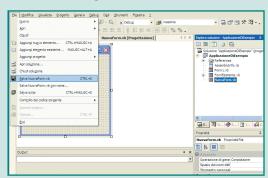
Paint - L'evento Paint viene generato immediatamente prima che la form diventi visibile ed ogni volta che la finestra viene disegnata, ad esempio quando viene ingrandita, ridotta ad icona o ripristinata. Nell'evento Paint vengono di norma eseguite operazioni di spostamento o di ridimensionamento dei controlli in una form di cui sono state modificate le dimensioni. VB .NET mette, comunque, a disposizione degli strumenti per il ridimensionamento automatico dei controlli.

Activated e Deactivate - L'evento *Activated* viene

La routine *InitializeComponent* viene utilizzata dall'ambiente di sviluppo per conservare i valori delle proprietà impostati nella finestra *Progettazione Windows Form.* In Visual Basic 6 queste informazioni non erano salvate come codice, ma come istruzioni in formato testo all'inizio del file *.FRM* e non venivano mai mostrate nella finestra del codice.

SALVARE UNA FORM

Dopo aver creato una nuova form è necessario salvarla form sull'hard disk. Per questo possiamo utilizzare quattro diversi metodi

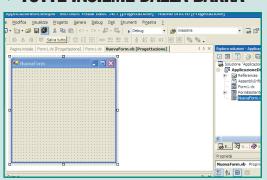
> DAL MENU FILE

1 Ciccando sulla voce Salva NuovaForm.vb, dove NuovaForm è il nome della form selezionata (CTRL+S).

> DALLA BARRA DEGLI STRUMENTI

2 Per accedere rapidamente cliccando sull'icona Salva NuovaForm.vb dalla barra degli strumenti.

> TUTTE INSIEME

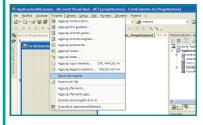
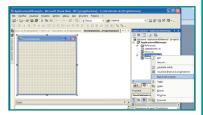
3 Se il progetto è composto da più form, selezionando dal menu File la voce Salva Tutto (CTRL+MAIUSC +S)

> TUTTE INSIEME DALLA BARRA

4 L'accesso rapido per salvare tutte le form insieme cliccando sull'icona Salva Tutto dalla barra degli strumenti.

generato nel momento in cui una form diventa la form attiva, subito dopo l'evento *Paint.* L'evento *Activated* può essere utilizzato quando si vogliono ripristinare situazioni che possono essere state modificate da un'altra form quando la form non era attiva. L'evento *Deactivate* viene generato quando viene attivato un'altra form dell'applicazione, per poi generare un altro evento *Activated* nel momento in cui lo stato attivo ritorna alla form di partenza.
Closing - L'evento *Closing* si verifica alla chiusura della form. L'evento può essere annullato, impostando la proprietà *Cancel,* passato al gestore eventi, su *true,* in questo caso la finestra rimane aperta. Questo evento viene usato per chiedere all'utente se si vogliono salvare i dati eventualmente modificati (per intenderci è quello che ci chiede *Word* ogni volta che stiamo per chiudere il programma). Al termine dell'evento *Closing* (sempre se non si è impostato *Cancel=True*) la form viene scaricata.
Closed - L'evento *Closed* si verifica quando la finestra è chiusa e non è più visibile. È possibile utilizzare questo evento per eseguire operazioni quali il salvataggio di informazioni immesse nella form o la liberazione di risorse utilizzate dalla form. Infine quando la form viene distrutta viene generato l'evento *Dispose.*

ELIMINARE UNA FORM DAL PROGETTO

Per eliminare una form dal progetto possiamo utilizzare due metodi diversi:

METODO 1
Possiamo selezionare la form che si vuole escludere dal progetto e, dal menu Progetto selezionare la voce Escludi dal progetto.

METODO 2
Possiamo cliccare con il tasto destro del mouse nella finestra Esplora Soluzioni sul form che si vuole eliminare e selezionare Escludi dal progetto

In tutti e due i modi, una form salvata in precedenza non viene cancellata fisicamente dall'Hard Disk. Per eliminare ogni traccia della form si devono usare gli strumenti di Windows e cancellare fisicamente il file con il nome della form, oppure si può selezionare la voce Elimina nella finestra Esplora Soluzioni

I METODI

Per rendere visibile una form, dobbiamo utilizzare il metodo *Show,* dopo aver creato un'istanza della form stessa, utilizzando la classica parola chiave *New* della programmazione ad oggetti. Il codice necessario a mostrare una finestra è il seguente:

```
Dim frm As New Form1()
frm.Show()
```

È possibile visualizzare una form in due modalità

- **Modal** - sono finestre a scelta obbligatoria che in genere impongono all'utente una risposta al loro quesito prima di restituire il controllo all'applicazione.
- **Modeless -** sono le finestre standard che non impongono nessuna scelta obbligatoria.

Per visualizzare una finestra non modale si deve usare il metodo *Show* (questo metodo non accetta argomenti).

```
frm.Show()
```

Per visualizzare una finestra a scelta obbligatoria si deve usare il metodo *Showdialog.*

```
frm.Showdialog()
```

Quando viene visualizzato una form a scelta obbligatoria, il codice successivo alla chiamata del metodo *Showdialog* non viene eseguito fino a quando la finestra visualizzata non viene chiusa, e l'input da tastiera o dal mouse è valido solo per gli oggetti della form. Viceversa nel caso di finestre modeless il codice successivo alla chiamata del metodo *show* viene eseguito progressivamente, La visualizzazione della nuova form non interrompe, quindi, il flusso di esecuzione del codice e l'utente può passare da questa a qualsiasi altra form dell'applicazione. Per nascondere temporaneamente una form si può utilizzare il metodo *Hide* che nasconde la form impostandone la proprietà *Visibile* a *False*

```
frm.Hide()
```

Nascondendo una form, però, la finestra rimane caricata in memoria. Per cancellare definitivamente dalla memoria una form dobbiamo utilizzare il metodo *Close*

```
frm.Close()
```

CONCLUSIONI

Tramite l'impostazione delle proprietà della form, il disegno dei controlli e la scrittura di codice VB per la risposta agli eventi, è possibile creare interfacce sempre più attraenti. Nei prossimi numeri analizzeremo i vari tipi di form provando a dare una risposta sul tipo giusto da usare in ogni occasione.

Luigi Buono

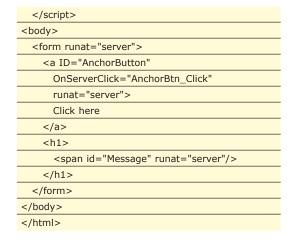
NOTA

Per creare una nuova form in fase di progettazione, si può anche: cliccare sull'icona corrispondente nella barra degli strumenti. Cliccare con il tasto destro del mouse sul progetto, nella finestra *Esplora soluzioni,* selezionare la voce *Aggiungi* e poi *Aggiungi Windows Form.*

I web controls di ASP.NET

Usare i web control, senza conoscerne da vicino le funzionalità, può essere un'operazione non priva di difficoltà. Con una guida come questa, districarsi tra tutte le possibilità diventa semplice

Districarsi tra un numero elevato di nuovi web controls, per chi è alle prime armi con ASP.NET, non è una pratica banale. I problemi cominciano non appena si passa, dalle prime applicazioni create per sperimentare, a quelle ben più complesse destinate ai sistemi di produzione. Scegliere tra un vasto insieme di controls può disorientare anche gli utenti più smaliziati, per tale motivo cercheremo di darne una classificazione ordinata.

GLI HTMLCONTROLS

La prima distinzione che si fa all'interno dei web controls prevede la suddivisione in due grandi famiglie, che prendono il nome dalla parte finale del namespace nel quale sono collocati. Gli *HtmlControls*, sono posizionati all'interno del namespace *System.Web.UI.HtmlControls,* ed i WebControls, che sono collocati nel namespace *System.Web.UI.WebControls*. Al primo gruppo appartengono, molto semplicemente, tutti i tag HTML a cui viene aggiunta la proprietà *runat= "server"*. Possiamo dire, banalmente, che è il sistema che ASP.NET li usa per mappare gli oggetti della pagina sulle istanze di classi che poi andrà a creare, in fase di compilazione. Ovviamente esistono diverse classi che prendono il nome dalla tipologia di tag HTML a cui si riferiscono, tra cui *HtmlAnchor,* che rappresenta il tag *<a />*, *HtmlTable* che rappresenta il tag *<table />* e così via. Un esempio classico di uso degli *HtmlControls* è il seguente:

```
<%@ Page Language="VB" AutoEventWireup="True" %>
<html>
  <script runat="server">
    Sub AnchorBtn_Click(sender As Object, e As
                                          EventArgs)
      Message.InnerHtml = "Hello World!!"
    End Sub
```

```
  </script>
<body>
  <form runat="server">
    <a ID="AnchorButton"
      OnServerClick="AnchorBtn_Click"
      runat="server">
      Click here
    </a>
    <h1>
      <span id="Message" runat="server"/>
    </h1>
  </form>
</body>
</html>
```

Tutti i controlli marcati come *Runat=Server* sono identificabili come *HtmlControls*. Gli *HtmlControls* godono spesso di particolari proprietà e metodi utilizzabili via codice. Per tutti gli altri tag, che non prevedono funzioni particolari o non sono utilizzati di frequente, ASP.NET prevede una classe *HtmlGenericControl* che consente di accedere a questi ultimi in maniera semplice, ma senza avere accesso alle rispettive proprietà HTML, come invece si può fare nel caso degli *HtmlControls* mappati direttamente. Eccone un esempio di utilizzo:

```
<a id="link" runat="server" />
```

Nel codice, per impostare le proprietà di questo elemento, faremo riferimento, semplicemente, a quelle che il tag HTML corrispondente offre abitualmente nella creazione di pagine HTML, ad esempio:

```
link.Href="http://www.aspitalia.com";
link.InnerText = "ASPItalia.com";
```

Questo approccio consente di mantenere le conoscenze acquisite in anni di utilizzo di HTML, potendo al tempo stesso sfruttare le caratteristiche di ASP .NET, che rendono possibile programmare lato ser-

ver i controls presenti sulle pagine, trattandoli come oggetti. È dunque spesso utilizzato da chi comincia ad utilizzare ASP.NET, perché è di sicuro più semplice da apprendere.

I WEBCONTROLS

A differenza degli *HTMLControls* i *WebControls* sono controlli complessi che non fanno direttamente riferimento ai tag html classici. Ad esempio:

```
<mytree:treeview runat="server">
  <mytree:treenode Text="Michigan">
    <mytree:treenode Text="Detroit" />
    <mytree:treenode Text="Farmington" />
  </mytree:treenode>
  <mytree:treenode Text="Washington" >
    <mytree:treenode Text="Bellevue" />
    <mytree:treenode Text="Redmond" />
  </mytree:treenode>
</mytree:treeview>
```

Questo Controllo disegna un albero, o una *treeview*, cosa che risulterebbe impossibile o molto complessa utilizzando i soli *HTMLControls*. Inoltre i *WebControls* essendo nati in modo specifico all'interno di un ambiente ad oggetti, a differenza dei loro predecessori, godono di alcune caratteristiche interessanti, come ad esempio: *"la coerenza dei nomi dei membri delle classi all'interno di un insieme omogeneo per funzionalità di controls"*. Ad esempio potremo stare tranquilli che la proprietà *Text* all'interno dei *WebControls* sarà sempre utilizzata per modificare il contenuto del testo di un oggetto che ne fa uso. E che questa proprietà si chiamerà sempre *"Text"* non per esempio *"Label", "Content"*, o altro. Il primo vero obiettivo dei *WebControls* è dunque quello di fornire, a chi li utilizza, un insieme di proprietà simili su tutta la famiglia. Un esempio di quanto detto è il seguente

```
<asp:label id="testo" runat="server" />
<asp:button id="pulsante" runat="server" />

testo.Text = "Questo è il testo";
pulsante.Text = "Questo è il testo";
```

Anche con controlli che offrono funzionalità davvero differenti tra di loro, come in questo caso, il modello ad oggetti condiviso permette di mantenere inalterato il modo di operare. Di particolare interesse, perché consentono di racchiudere al proprio interno altri controls, sono i controlli *Panel* e *Placeholder*. Concettualmente sono identici, ma *Panel* aggiunge un "blocco" interno ai tag, che nel caso di browser uplevel viene convertito in un tag *<div />*, negli altri casi una *<table />*, mentre *Placeholder* si limita ad includere al proprio interno i controls. Entrambi sono utili quando si vogliono nascondere pezzi della pagina, sfruttando la proprietà *Visible*, ad esempio nel caso in cui la pagina sia composta da un wizard che consente di costruire passo-passo il risultato finale.

RICH CONTROLS E DATA CONTROLS

Del secondo gruppo abbiamo già parlato a sufficienza nelle precedenti puntate di questa serie, trattando l'accesso ai dati con ADO.NET. A questa famiglia appartengono i già noti *DataGrid, DataList* e *Repeater* ed un'ulteriore sottofamiglia di controls, i *List Controls*. Di questo gruppo fanno parte *DropDownList, ListBox, CheckBoxList* e *RadioButtonList*. Come il nome suggerisce, si tratta di controls che permettono di implementare semplicemente liste di opzioni, rispettivamente attraverso un elenco a scelta, uno a scelta multipla, un elenco di checkbox ed uno di radio button. Tutti questi controls possono ricevere la lista dei propri valori da un datasource, proprio come un *Data Control*, piuttosto che attraverso la definizione di sottocontrolli, chiamati *ListItem*. Vediamo un esempio che ci permette di capire quanto il modello unificato sia vantaggioso.

Creiamo un semplice elenco, che attraverso una *DropDownList* mostri una serie di colori da poter visualizzare, con il colore selezionato dall'utente mostrato attraverso una Label:

Fig. 1: ***Cliccando sulla combobox viene modificato il contenuto dell messaggio***

```
<form runat="server">
<p>Seleziona un valore:
<asp:dropdownlist id="data" runat="server"
                              autoPostBack="true">
<asp:listitem>Rosso</asp:listitem>
<asp:listitem>Nero</asp:listitem>
<asp:listitem>Verde</asp:listitem>
</asp:dropdownlist>
</p>
<p>Hai selezionato:
<asp:label id="testo" runat="server" />
</p>
</form>
<SCRIPT RUNAT="SERVER">
void Page_Load() {
```

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

```
   if (Page.IsPostBack) {
      testo.Text = data.SelectedValue;} }
</SCRIPT>
```

Il risultato è visibile nell'immagine sottostante e mostra il nostro elenco di opzioni con la relativa selezione da parte dell'utente. Se vogliamo cambiare il *List Control,* ci basta sostuire le occorrenze di *DropDownList* con il rispettivo control da utilizzare. Nel nostro caso optiamo per un elenco di radio button, attraverso *RadioButtonList*. L'effetto che si ottiene è mostrato in **Figura 2**; quello che abbiamo fatto è stato semplicemente modificare la pagina nella definizione del solo controls. Né il codice, né tanto meno l'elenco dei valori sono stati modificati. Ai *rich controls,* infine, appartiene un insieme di oggetti che, mediamente, aggiungono funzionalità complesse. Il più famoso di tutti è sicuramente *Calendar,* che aggiunge un calendario completo semplicemente inserendo il codice *<asp:calendar runat= "server"* /> nella pagina. Gli altri due che mancano all'appello sono *AdRotator,* che permette di far ruotare banner, e Xml, che invece consente di mostrare facilmente nella pagina il frutto di trasformazioni XSLT.

Fig. 2: *Per ottenere un layout completamente diverso è stato sufficiente sostituire il controls combobox con uno di tipo RadioButtonList*

I VALIDATOR CONTROLS

Uniti dal fatto di ereditare tutti dalla classe *BaseValidator,* i *Validator Controls* sono accomunati ovviamente per le funzionalità che espongono, ovvero dalla possibilità di convalidare l'input dell'utente all'interno di una web form sfruttando un modello ad oggetti condiviso. Ogni validator espone comunque differenze dovute alla diversa tipologia di validazione che rende disponibile all'utente, con un buon nucleo delle funzionalità esposte condivise all'interno della famiglia:

- **ControlToValidate:** indica il nome del control sul quale applicare la validazione;
- **Display:** indica la modalità in cui il messaggio di errore deve essere visualizzato.
  - **None:** non visualizza l'errore;
  - **Static:** visualizza l'errore, occupando subito lo spazio necessario nella pagina;
  - **Dynamic:** visualizza l'errore, ma lo spazio nella pagina viene occupato nel caso debba essere mostrato;
- **EnableClientScript:** attiva il controllo lato client;
- **Enabled:** attiva la convalida;
- **ErrorMessage:** il messaggio da visualizzare in un *ValidationSummary*;
- **IsValid:** contiene un boolean con il risultato della validazione;
- **Text:** specifica il testo da visualizzare in caso di errore.

Ovviamente ci sono diversi tipi di validazione e per ciascun tipo è presente un control dedicato. Nei casi in cui non sia possibile sfruttare i controlli già presenti, se ne possono costruire di propri, sfruttando il control *CustomValidator.* Da segnalare, infine, che a tutti i validator manca la possibilità di controllare che l'input sia presente, dunque se utilizzate, ad esempio, un *RangeValidator* è necessario associare al campo sul quale deve essere fatta la convalida anche un *RequiredFieldValidator.* Perché la convalida possa funzionare, bisogna associare, al validator il control su cui effetuare la validazione attraverso la proprietà *ControlToValidate.* È da sottolineare il fatto che la convalida funziona sia lato client che lato server, ma solo ed esclusivamente, nella versione 1.1, con Internet Explorer 6.0 o successivi. Questa limitazione è dovuta al fatto che le definizioni dei browser di ASP.NET 1.x non sono aggiornate e che, in fase di progettazione, è stata fatta la scelta di utilizzare codice Javascript non compatibile con il DOM, ma sono con IE. Fortunatamente questo limite non ci sarà più con la versione 2.0, in arrivo per fine anno. In tutti gli altri casi (ad esempio usando FireFox) la convalida avviene solo lato server, garantendo che comunque l'input inserito dall'utente sia conforme a quanto specificato in fase di creazione dell'applicazione.

UN ESEMPIO PRATICO

Per apprezzare al meglio l'utilità dei *Validator Controls,* costruiamo con pochi semplici passaggi una form tipo per la registrazione di un utente nel database:

1 Partiamo con la defizione di un campo nel quale inserire il nome dell'utente, associando un *RequiredFieldValidator* per essere sicuri che l'utente non dimentichi di inserire il nome:

```
Inserisci il tuo nome: <asp:textbox id="nome"
                                   runat="server"/>
<asp:RequiredFieldValidator runat="server"
                            ControlToValidate="nome"
```

```
    ErrorMessage="* devi specificare il tuo nome"
                                   Display="static"/><br />
```

2 Continuiamo con la definizione di due campi per l'inserimento dell'indirizzo e-mail, in modo da evitare che l'utente, per sbaglio, non ne specifichi uno errato. In questo caso utilizziamo un *CompareValidator* tra i due controlli, specificando come valore dell'attributo *Operator* la stringa *"Equal"*:

```
Il tuo indirizzo e-mail: <asp:textbox id="email1"
                                          runat="server"/>
<asp:RequiredFieldValidator runat="server"
                              ControlToValidate="email1"
    ErrorMessage="* devi specificare il tuo indirizzo"
                                   Display="static"/><br />
Ripeti il tuo indirizzo email: <asp:textbox
                        id="email2" runat="server"/><br />
<asp:CompareValidator runat="server"
    ControlToValidate="email1"
                              ControlToCompare="email2"
    Operator="Equal"
    ErrorMessage="* i valori inseriti devono combaciare"
    Display="dynamic" /><br />
```

3 Proseguiamo quindi con la definizione di un campo nel quale inserire una data (ad esempio, di nascita) e sfruttando il *RangeValidator,* andiamo a verificare che sia superiore al primo gennaio 1900:

```
Inserisci una data successiva al 1900:
<asp:textbox id="data" runat="server"/>
<asp:RangeValidator runat="server"
    ControlToValidate="data"
    MaximumValue="1/1/2900"
    MinimumValue="1/1/1900"
    Type="Date"
    ErrorMessage="* La data deve essere successiva
            al 01/01/1900" Display="static" /><br />
```

4 Infine inseriamo un ultimo campo per la password e sfruttando il *RegularExpressionValidator,* permettiamo l'inserimento di caratteri alfanumerici (minimo 1, massimo 10):

```
Inserisci la password (solo caratteri alfanumerici,
                                    max 10 caratteri):
<asp:textbox id="password" runat="server"/>
<asp:RequiredFieldValidator runat="server"
                            ControlToValidate="password"
     ErrorMessage="* devi inserire un valore"
                                    Display="dynamic"/>
<asp:RegularExpressionValidator id="valtextbo1_req"
                                         runat="server"
    ControlToValidate="password"
    ValidationExpression="\w{1,10}"
    ErrorMessage="* massimo 10 caratteri alfanumerici"
    display="dynamic" /><br />
```

5 A questo punto aggiungiamo un pulsante sulla pagina, alla cui pressione andiamo a verificare che *Page.IsValid* sia vero.

```
Sub ValidaForm(sender As Object, e As
                                     System.EventArgs)
    lblText.Text = String.Empty
    if Page.IsValid then
        lblText.Text = "Pagina valida!"
    end if
End Sub
```

È sempre importante verificare questa proprietà lato server, perché nel caso in cui la convalida lato client fosse disattivata, è il vero segnale che il controllo è andato a buon fine.

CONCLUSIONI

Scegliere nel modo più appropriato il control da utilizzare è un punto molto importante nello sviluppo di applicazioni web basate su ASP.NET, perché una scelta errata può portare a perdere molto più tempo di quanto una scelta consapevole possa invece aiutare. Mettendo in pratica le semplici istruzioni di questo articolo scegliere il control più adatto alle proprie esigenze diventa un compito più agevole da sopportare. Aggiungere controlli di validazione: passo-passo

Daniele Bochicchio

L'AUTORE

Daniele Bochicchio è il content manager di *ASPItalia.com,* community che si occupa di ASP.NET, Classic ASP e Windows Server System. Il suo lavoro è principalmente di consulenza e formazione, specie su ASP.NET, e scrive per diverse riviste e siti. È Microsoft ASP.NET MVP, un riconoscimento per il suo impegno a supporto delle community e per l'esperienza maturata negli anni. Il suo blog è all'indirizzo http://blogs.aspitalia.com/daniele/

Nome	Codice	Descrizione
Button	<asp:Button ID="bottone1" text="Cliccami" CommandName= "Funzione" CommandArgument= "Argomenti" runat="server" />	**Aggiunge un pulsante (<input type="button">) alla cui pressione viene invocato un evento server side.**
CheckBox	<asp:CheckBox ID="chk1"text= "Selezionami" checked= "true"runat="server" />	**Inserisce una checkbox.**
HyperLink	<asp:HyperLink ID="link1" Text ="Clicca su questo link" NavigateUrl="http://www.aspitalia.com" runat="server" />	**Inserisce un link con una descrizione presa dalla proprietà Text, che punta all'URL specificato in NavigateUrl.**
Image	<asp:Image ID="img1"ImageUrl ="img.gif"ToolTip="Testo dell'alt"width="80"height="30"runat="server" />	**Aggiunge un'immagine alla pagina, con **
ImageButton	<asp:ImageButton ID="imgbtn1"ImageUrl ="img.gif" ToolTip= "Testo dell'alt"runat="server" />	**Aggiunge un pulsante con un'immagine cliccabile. Produce <input type= "image" />**
Label	<asp:Label ID="lbl1"Text="testo del controllo" CssClass= "classe" runat="server" />	**Aggiunge alla pagina il testo specificatonella proprietà Text.**

Tabella 1: Principali Webcontrol e la loro descrizione

Usare gli Array alle basi del codice!

Diremo qualcosa su una delle strutture dati che costituisce le fondamenta di ogni linguaggio di programmazione, impareremo come utilizzarla all'interno di JavaScript e vedremo cosa il linguaggio ci offre

REQUISITI

Conoscenze richieste
Basi di Javascript

Software
-

Impegno

Tempo di realizzazione

La logica che sta alla base degli array è molto semplice. Si tratta di collezioni di variabili.
L'immagine classica di un Array è quella di un treno, dove tutte le carrozze sono numerate. Il contenuto di ciascuna carrozza è accessibile con una sintassi del tipo *treno[1], treno[2]* etc... È anche vero che riferendoci all'insieme dei treni, potremmo scrivere qualcosa del tipo *treno[1][1], treno[1][2], treno[2][1],treno[2][2],* per riferirci rispettivamente alla prima e seconda carrozza del treno *uno* e la prima e la seconda carrozza del treno *due*. In sostanza un *array* è una struttura formata da un insieme di elementi, reperibili attraverso uno o più indici. Il numero di indici permette di definire la dimensione dell'array.
Il numero di elementi dell'array permette di definire la lunghezza dello stesso. In javascript un *array* viene generato creando una istanza della classe *Array()* nel seguente modo:

```
<script type="text/javascript">
var Frutta = new Array(3);
Frutta[0] = "Mela";
Frutta[1] = "Pera";
Frutta[2] = "Banana"
</script>
```

Si noti che gli indici degli array iniziano la numerazione a partire da zero. Inoltre, nella dichiarazione della dimensione di un array si usano le parentesi tonde, mentre invece nella valorizzazione si usano le parentesi quadre. Occhio alle parentesi, è un errore comune scambiarle !
Per un Array, è definita la sola proprietà: *length* che restituisce il numero di elementi di cui è composto l'Array. Un esempio d'utilizzo della proprietà è il seguente:

```
for(i=0;i<=Frutta.length-1;i++)
{
   document.write("Frutta[" + i +"] = " + Frutta[i] +
                                              "<br>");
}
```

L'output dello script è la stampa di tutti gli elementi dell'array. Si noti che nel ciclo, si è considerato l'indice i che parte da zero, quindi per poter essere sicuri di ciclare su tutti gli elementi dell'array occorre proseguire sino ad *Frutta .length-1,* e non sino a *Frutta.length*. Anche questo è un errore comune... La classe *Array* in Javascript mette a disposizione alcuni metodi per la loro gestione, in particolare quelli elencati in **Tabella 1**.
Tanto per chiarire, facciamo qualche esempio d'uso dei vari metodi:

```
var Frutta = new Array("Mela", "Pera", "Limone");
```

COME INIZIARE

Per provare gli script proposti nell'articolo non avete bisogno di molti strumenti. Prima di tutto vi occorre un editor di testo, il notepad andrà benissimo. Se volete avere invece qualche comodità potete usare un editor evoluto, come ad esempio DreamWeaver, o simili. Chiaramente i file html che andremo a generare devono essere salvati in una directory sotto un web server per potere essere eseguiti. Potere installare Apache in locale prendendolo dal cd allegato alla rivista, oppure andrà benissimo IIS se siete in ambiente Windows, oppure potete provare gli script presso il vostro provider di hosting. Di seguito riportiamo lo scheletro di uno script JavaScript, potete tranquillamente utilizzarlo come base per il testing delle vostre pagine HTML, dovete semplicemente creare un file di testo con queste istruzioni dentro e salvarlo come nomefile.html in una directory del vostro Web Server

```
<hea@d>
<script type="text/javascript">
</script>
</head>
<body>
</body>
</html>
</come iniziare>
```

```
document.write("Frutta.toString() = "
                        + Frutta.toString());
// Restituisce la stringa: Mela,Pera,Limone
```

Nella dichiarazione dell'array, si è utilizzato un metodo alternativo che permette sia la dichiarazione che la valorizzazione dell'array in un passo unico. Il metodo *toString()* restituisce semplicemente gli elementi dell'array concatenati dalla virgola.

```
document.write("Frutta.concat(\" Un po'di frutta
        \",\"Italiana!\") = " + Frutta.concat(" Un po'di
                        frutta","Italiana!"));
//Restituisce la stringa: Mela,Pera,Limone,
                        Un po' di frutta,Italiana!
```

Il metodo *concat(x1,x2,..)* concatena agli elementi dell'array gli elementi che vengono passati come input al metodo. Tale metodo è utile nel caso di due array. Ad esempio:

```
var Frutta = new Array("Mela", "Pera", "Limone");
var FruttaTropicale = new Array("Jambulo", "Cocco",
                        "Mango");
TuttaFrutta = Frutta.concat(FruttaTropicale);
document.write("TuttaFrutta.toString() = "
                        + TuttaFrutta.toString());
// Restituisce la stringa: Mela,Pera,Limone, Jambulo,
                        Cocco, Mango.
```

Nella terza riga del codice, usando *concat()* con due array, si è creato un nuovo array *TuttaFrutta()* composto dagli elementi dell'array *Frutta()* e dell'array *FruttaTropicale()*. Si noti che ne l'array *Frutta()* ne l'array *FruttaTropicale()* sono stati modificati.

```
var Frutta = new Array("Mela", "Pera", "Limone");
strFrutta_01 = Frutta.join(" *** ")
document.write("strFrutta = " + strFrutta);
// Restituisce Mela *** Pera *** Limone
strFrutta_02 = Frutta.join()
document.write("strFrutta = " + strFrutta);
// Restituisce Mela,Pera,Limone
```

Il metodo *join()* viene utilizzato per riunire in una stringa gli elementi di un array. Crea una stringa di caratteri composta dagli elementi dell'array separati dal carattere che è passato in input al metodo.
Nel caso non si passino caratteri, per default è utilizzato il carattere virgola (e funziona come *toString()*). È l'opposto del metodo *split()* che analizzeremo parlando della classe *String()*.

```
var Frutta = new Array("Mela", "Pera", "Limone");
document.write("Frutta.pop() = " + Frutta.pop());
```

#	Metodo	Descrizione
1	*toString()*	Restituisce una stringa composta dagli elementi dell'array
2	*concat(x1,x2,..)*	Concatena gli elementi *x1, x2,..* con l'array cui è applicato il metodo. Tipicamente *x1,x2,..* sono array.
3	*join(chrseparatore)*	Converte in stringa gli elementi dell'array e li separa con il carattere chrseparatore, che è facoltativo.
4	*pop()*	Rimuove l'ultimo elemento dell'array e lo restituisce come output del metodo
5	*push(x1,x2,..)*	Aggiunge *x1, x2,..* in coda all'array, nell'ordine in cui essi compaiono
6	*reverse()*	Ribalta l'ordine con cui sono sistemati gli elementi dell'array
7	*shift()*	È la funzione duale di *pop()*. Rimuove il primo elemento dell'array e lo restituisce come output del metodo.
8	*unshift(x1,x2,..)*	È la funzione duale di *push()*. Aggiunge *x1, x2,..* in testa all'array, nell'ordine in cui essi compaiono
9	*slice(start,end)*	Restituisce un array con gli elementi compresi tra *start* (compreso) ed *end* (quest'ultimo non incluso)
10	*sort(comparefn)*	Esegue l'ordinamento degli elementi dell'array, utilizzando come criterio di confronto la funzione comparefn in input.
11	*splice(start, deleteCount,x1,x2,..)*	Rimuove tanti elementi quanti specificati dal parametro *deleteCount* a partire dall'elemento start compreso, e li sostituisce con gli elementi *x1,x2,..* se presenti, nell'ordine in cui compaiono.
12	*valueOf()*	Ritorna il valore primitivo dell'oggetto *Array()*

Tabella 1: Un elenco dei metodi per la gestione degli array

```
// Restitituisce: Limone
for(i=0;i<=Frutta.length-1;i++)
{
  document.write("Frutta[" + i +"] = " + Frutta[i] +
                        "<br>");
}
// Stampa solo i due elementi "Mela" e "Pera".
```

Il metodo *pop()* elimina l'ultimo elemento dall'array e lo restituisce come output. Il risultato è quindi che l'array *Frutta()*, dopo che ad esso è stato applicato il metodo *pop()*, si trova senza l'elemento *"Limone"*.

```
var Frutta = new Array("Mela", "Pera", "Limone");
Frutta.push("Fragola","Albicocca");
for(i=0;i<=Frutta.length-1;i++)
{
  document.write("Frutta[" + i +"] = " + Frutta[i] +
                        "<br>");
}
```

Il metodo *push()* aggiunge in coda all'array le stringhe che riceve in input. In pratica, adesso l'array *Frutta()* contiene due elementi in più che sono *"Fragola"* ed *"Albicocca"*. Restituisce la lunghezza dell'array, nel nostro caso *Frutta.push("Fragola","Albicocca")* restituisce 5. Attenzione a non utilizzare *push()* per concatenare due array; non funziona. Un codice del tipo:

```
var Frutta = new Array("Mela", "Pera", "Limone");
var FruttaTropicale = new Array("Jambulo", "Cocco",
                        "Mango");
Frutta.push(FruttaTropicale);
for(i=0;i<=Frutta.length-1;i++)
{
  document.write("Frutta[" + i +"] = " + Frutta[i] +
```

Utilizza questo spazio per le tue annotazioni

```
                                  "<br>");
}
```

Restituisce quanto segue:

```
Frutta[0] = Mela
Frutta[1] = Pera
Frutta[2] = Limone
Frutta[3] = Jambulo, Cocco, Mango;
```

In pratica, javascript applica per default il metodo *toString()* all'array che viene passato come argomento a *push()*. Quindi il codice *Frutta .push (FruttaTropicale)* equivale a *Frutta.push(FruttaTropicale.toString());*

```
var Frutta = new Array("Mela", "Pera", "Limone");
Frutta.reverse();
for(i=0;i<=Frutta.length-1;i++) {
   document.write("Frutta[" + i +"] = " + Frutta[i] +
                                  "<br>");
}// Ritorna un array i cui elementi sono nell'ordine:
                            Limone, Pera, Mela
```

Il metodo *reverse()* non fa altro che ribaltare l'ordine degli elementi dell'array. Il primo elemento diventa l'ultimo, l'ultimo il primo e via di seguito con quelli intermedi.

```
var Frutta = new Array("Mela", "Pera", "Limone");
var strShift = Frutta.shift();
document.write("Frutta.shift() = " + strShift);
//Ritorna "Mela"
for(i=0;i<=Frutta.length-1;i++)
{
   document.write("Frutta[" + i +"] = " + Frutta[i] +
                                  "<br>");
}// Ritorna un array i cui elementi sono nell'ordine:
                                 Pera , Limone.
```

Il metodo *shift()* elimina il primo elemento dell'array e lo restituisce come output. È il duale di *pop()*.

```
var Frutta = new Array("Mela", "Pera", "Limone");
Frutta.unshift("Mora","Albicocca");
for(i=0;i<=Frutta.length-1;i++)
{
   document.write("Frutta[" + i +"] = " +
                              Frutta[i] + "<br>");
}// Ritorna un array i cui elementi sono nell'ordine:
                 Mora, Albicocca,Mela, Pera, Limone
```

Il metodo *unshift(x1,x2,..)* aggiunge in coda all'array le stringhe che vengono passate in input. È il metodo duale di *push(x1,x2,..)*. Il metodo *unshift(x1,x2,..)*, come *push(x1,x2,..)* non è utilizzabile per concatenare array.

```
var Frutta = new Array("Mela", "Pera",
                     "Limone","Mora","Albicocca");
var FruttaSlice = Frutta.slice(1,3);
// Stampa l'array FruttaSlice
for(i=0;i<=FruttaSlice.length-1;i++)
{
   document.write("FruttaSlice[" + i +"] = " +
                           FruttaSlice[i] + "<br>");
}
// Stampa l'array FruttaSlice, nel quale è stato
                             omesso il valore "end"
var FruttaSlice = Frutta.slice(3);
for(i=0;i<=FruttaSlice.length-1;i++)
{
   document.write("FruttaSlice[" + i +"] = " +
                           FruttaSlice[i] + "<br>");
}
```

L'espresione *Frutta.slice(1,3)* restituisce un array i cui elementi sono gli elementi 1 e 2 dell'array *Frutta()*, ossia *"Pera"* e *"Limone"*. Quindi l'elemento corrispondente al valore *start* è compreso, mentre quello corrispondente all'indice *end* è escluso. L'array originario *Frutta()* rimane inalterato.

```
var Frutta = new Array("Mela", "Pera",
                     "Limone","Mora","Albicocca");
// Stampo l'array Frutta() originale
for(i=0;i<=Frutta.length-1;i++)
{
   document.write("Frutta[" + i +"] = " + Frutta[i] +
                                  "<br>");
}
// Ordino l'array Frutta() e lo stampo.
Frutta.sort();
for(i=0;i<=Frutta.length-1;i++)
{
   document.write("Frutta[" + i +"] = " + Frutta[i] +
                                  "<br>");
}
```

Il codice sopra riportato rappresenta il primo esempio di utilizzo del metodo *sort()*, senza parametri in input. Il metodo provvede ad ordinare gli elementi delll'array *Frutta()*, che di conseguenza viene modificato in un nuovo array contenente gli stessi elementi dell'array originale, ma con un ordinamento differente. Si noti che è possibile utilizzare direttamente il metodo, senza assegnarlo ad una variabile. In altri termini, entrambi le espressioni sono corrette:

```
Frutta.sort();
var FruttaSort = Frutta.sort();
```

La prima espressione ordina l'array *Frutta()*, la seconda espressione, oltre a ordinare l'array

Frutta(), crea il nuovo array *FruttaSort()*, duplicato di *Frutta()*, con gli elementi in ordine alfabetico. Che cosa succede se applichiamo il metodo *sort()* ad un array di numeri ?

```
var Numeri = new Array(45,73,500,740,23);
// Stampo l'array Numeri() originale
for(i=0;i<=Numeri.length-1;i++)
{
        document.write("Numeri[" + i +"] = "
                        + Numeri[i] + "<br>");
}
// Ordino l'array Numeri() e lo stampo.
Numeri.sort();
for(i=0;i<=Numeri.length-1;i++)
{
  document.write("Numeri[" + i +"] = " + Numeri[i]
                                    + "<br>");
}
```

L'array di numeri "ordinato" stampa in questo caso la sequenza di numeri: *(23,45,500 ,73,740);* è immediato verificare tale sequenza non è ordinata secondo l'ordine numerico. Questo perché l'ordinamento utilizzato dal metodo *sort()* è sempre e solo alfabetico, e con questo criterio, 500 è "minore" di 73. Per fare in modo di ottenere un *sort()* corretto è il seguente:

```
var Numeri = new Array(45,73,500,740,23);
// Ordino l'array Numeri() e lo stampo.
Numeri.sort(ordina);
for(i=0;i<=Numeri.length-1;i++)
{
  document.write("Numeri[" + i +"] = " + Numeri[i]
                                    + "<br>");
}
function ordina(x1,x2)
{
  return (x1-x2);
}
```

In questo caso, la funzione *ordina(x1,x2)* viene utilizzata come criterio ordinamento per i numeri *x1* ed *x2* che viene passata in input, ordinando in senso crescente la sequenza. Scambiando *x1* ed *x2,* si ottiene invece un ordinamento decrescente. Altri approfondimenti si trovano nel box a lato pagina.

```
var Frutta = new Array("Mela", "Pera",
                    "Limone","Mora","Albicocca");
// Stampo l'array Frutta() originale
for(i=0;i<=Frutta.length-1;i++)
{
  document.write("Frutta[" + i +"] = " + Frutta[i] +
                                    "<br>");
}
// Applico il metodo splice()
Frutta.splice(2,2,"Cane","Gatto","Topo");
// Stampo l'array Frutta modificato.
for(i=0;i<=Frutta.length-1;i++)
{
  document.write("Frutta[" + i +"] = " + Frutta[i] +
                                    "<br>");
}
```

Il metodo *Frutta.splice(2,2,"Cane","Gatto","Topo")* nell'esempio sopra riportato, elimina due elementi a partire dal secondo incluso, ossia *"Limone"* e *"Mora"*, e li sostituisce con *"Cane"*, *"Gatto"*, *"Topo"*. L'array che ne deriva è:

```
Frutta[0] = Mela
Frutta[1] = Pera
Frutta[2] = Cane
Frutta[3] = Gatto
Frutta[4] = Topo
Frutta[5] = Albicocca
```

Un po' strana come frutta !
Veniamo ora la metodo *valueOf()*. La classe *Array()* non ha un implementato un metodo specifico per *valueOf()*, ma utilizza quello della classe padre *Object()*. Tale metodo si comporta, dal lato pratico, come *toString()* ed è poco utile con la classe *Array()*. Vedremo come meglio utilizzarlo con la definizione di funzione, in articoli successivi.
Un cenno agli array a più dimensioni. La dichiarazione:

```
var Frutta = new Array(3);
var Frutta[0] = new Array(4);
var Frutta[1] = new Array(4);
var Frutta[2] = new Array(4);
```

Permette la costruzione di un array *Frutta* a due indici con i seguenti elementi:

```
Frutta[0][0], Frutta[0][1], Frutta[0][2], Frutta[0][3]
Frutta[1][0], Frutta[1][1], Frutta[1][2], Frutta[1][3],
Frutta[2][0], Frutta[2][1], Frutta[2][2], Frutta[2][3]
```

Che possono essere gestiti come elementi di un array normale.

CONCLUSIONI

Gli array rappresentano uno dei pilastri della programmazione. Comprendere i metodi messi a disposizione dalla classe JavaScript che li gestice è fondamentale per scrivere applicazioni di qualunque tipo.

Danilo Berta

BlueTooth il re della comunicazione

Sfruttiamo il Nokia SDK per creare applicazioni che dialogano fra loro Un'introduzione ai servizi disponibili per altri telefonini e come cercare e connettersi a cellulari vicini

Il sempre crescente bisogno di comunicare ha portato, negli ultimi anni, alla diffusione di apparecchi telefonici portatili. Ciò che prima sembrava un privilegio di pochi è adesso accessibile a tutti.
Quotidianamente, infatti, premiamo freneticamente le dita su tastierini numerici di diverse forme e dimensioni. Se da una parte il mercato vede l'utente finale come un "utilizzatore", l'altra faccia della medaglia è occupata dagli sviluppatori di software, un laborioso sciame di api che impiega il suo tempo per creare prodotti di un certo livello. Sebbene Java, con i suoi difetti e le sue limitazioni, ha facilitato lo sviluppo di applicazioni, Symbian OS ci offre la possibilità di programmare un dispositivo mobile usando un linguaggio di più basso livello come il C. Nei precedenti articoli non sono state analizzate solo le tecniche di base, ma anche alcuni punti chiave del sistema come l'interfaccia grafica e la gestione dei contatti. Per completare il quadro non si può non rivolgere lo sguardo a come le informazioni entrano ed escono da un dispositivo. Sebbene esistano diversi standard, come il cavo seriale o i segnali a infrarossi, punteremo su quella strana parolina, a volte pronunciata male, che sentiamo dire tutti i giorni: bluetooth.

REQUISITI

Conoscenze richieste
Basi di C++, basi di Symbian C++

Software
Visual C++ 6 o sup. ActivePerl, Nokia SDK 1.2 o sup.

Impegno

Tempo di realizzazione

IL BLUETOOTH

Quando parliamo di bluetooth non dobbiamo pensare subito ad un qualcosa di fisico, bensì a "come" le informazioni vengono trasferite attraverso una rete senza fili. Il mezzo di comunicazione è essenzialmente lo stesso del wi-fi, ovvero le onde radio. Ciò che varia sono le frequenze e le regole per la trasmissione, che consentono di ottenere delle prestazioni migliori e delle velocità più elevate, limitando però le distanze. La comunicazione tra dispositivi bluetooth di classe *A* permette una comunicazione nel raggio di cento metri, massima distanza usufruibile da questa tecnologia. Questo è il motivo principale per cui bluetooth è particolarmente adatto solo per alcune situazioni, come il trasferimento di dati e il gioco in multiplayer tra dispositivi portatili. Inoltre l'adozione del bluetooth su dispositivi fissi come il personal computer, la tv e persino l'automobile, ha creato nuove opportunità per i più svariati usi.

UN INSIEME DI RUOTE DENTATE

Lo stack bluetooth, ovvero l'insieme di componenti di cui è composta l'architettura, è schematizzato in **Figura 1**.
I termini qui presenti saranno molto utili per la comprensione del codice. Il blocco inferiore riguarda la parte hardware e non verrà trattata, anche in virtù del fatto che le applicazioni non hanno accesso diretto a questo livello. Sarà proprio il sistema operativo Symbian che, grazie ad un ricco set di API, permetterà di usufruire dei benefici della tecnologia bluetooth. I singoli blocchi hanno ovviamente ruoli e caratteristiche diverse tra loro. Il protocollo RFCOMM, posizionato in cima al blocco superiore, consente di trattare la comunicazione bluetooth come se fosse una comunicazione seriale. Ciò ritorna utile in presenza di applicazioni legacy, ovvero datate. SDP (Service Discovery Protocol) riveste un ruolo fondamentale nella gestione dei servizi presenti sulla rete. Ogni dispositivo bluetooth possiede il proprio SDP. Esso avrà il compito di rendere disponibili, a chi lo richiede, tutte le informazioni utili per l'accesso ad uno specifico servizio, come l'indirizzo e il numero di porta.

Fig. 1: *Uno schema della composizione dello stack bluetooth*

Terminata la fase iniziale, SDP lascia subentrare L2CAP (Logical Link Controller and Adaptation Protocol). È questo il momento in cui le informazioni vengono pacchettizzate e ne viene effettuato il multiplexing sul canale. Tale livello è completamente trasparente al programmatore, in quanto implementato dal sistema operativo stesso. Per finire la parte più bassa dello stack software è occupata dal driver HCI (Host Controller Interface) che si preoccupa del dialogo con la parte hardware dello stack.

PUBBLICAZIONE DI UN SERVIZIO

Come appena accennato, il gestore SDP di ogni dispositivo deve permettere la registrazione di tutti i servizi presenti sul dispositivo stesso, affinché possano essere visti dall'intera rete. Symbian OS implementa quindi il database SDP come un server, in grado di dialogare con più processi contemporaneamente. La prima cosa da fare è ottenere un accesso a tale database. Le classi interessate sono due: RSdp e RSdpDatabase. Entrambe sono definite in btsdp.h.

```
RSdp database;
RSdpDatabase sessione;
if (database.Connect() != KErrNone) {
User::Panic(_L("ConnessioneBluetooth"),-1);
}
if (sessione.Open(database) != KErrNone) {
User::Panic(_L("ConnessioneBluetooth"),-1);
}
```

Una volta aperta la sessione con il database SDP si può aggiungere un servizio. Ricordiamoci di effettuare una chiamata close() su entrambi gli oggetti al termine dell'operazione, consentendo al sistema operativo di liberare le risorse occupate. Un servizio è rappresentato da un oggetto della classe *TSdpServRecordHandle* e la sua aggiunta al database SDP è effettuata richiamando la funzione *CreateServiceRecordL* sulla sessione ottenuta. Ecco il codice:

```
TSdpServRecordHandle record;
sessione.CreateServiceRecordL(KSerialClassID, record);
```

KSerialClassID è una costante che identifica il tipo di servizio. Volendo utilizzare una connessione seriale tramite il protocollo RFCOMM, semplice per la trattazione, il suo valore sarà 0x1101. Un elenco completo è consultabile al seguente url: *https://www.bluetooth.org/foundry/assignnumb/document/service_discovery*. Affinché il record sia attivo dobbiamo dichiararne esplicitamente alcuni attributi. Ciò viene fatto tramite la classe *CSdpAttrValueDES*, vediamo come:

```
CSdpAttrValueDES* protocolDescriptorList =
                    SdpAttrValueDES::NewDESL(NULL);
CleanupStack::PushL(protocolDescriptorList);
```

La comunicazione seriale è quel sistema che viene usato ancora oggi per il trasferimento di dati tramite un cavo ed una porta logica, che forse conosciamo come porta COM. I driver bluetooth che installiamo sui nostri pc utilizzano il protocollo seriale per interfacciare i vari software coi i dispositivi connessi alla rete.

COME INIZIARE

1 **Installiamo gli strumenti che ci servono: Nokia SDK 1.2, Microsoft Visual C++ 6.0 e ActivePerl. Durante l'installazione di Visual C++ ricordiamoci di settare le variabili d'ambiente. Il Nokia SDK può essere scaricato all'indirizzo http://www.forum.nokia.com/main/o,,034-4,00.html.**

2 **Aggiungiamo a Visual C++ il template Nokia per la creazione assistita di un'applicazione base. Copiamo *avkonappwiz.awx* e *avkonappwiz.hlp* da *Symbian\6.1\Series60\Series60Tools\applicationwizard* a *Microsoft Visual Studio \Common \MSDev98\Template*.**

3 **Apriamo Visual C++ e clicchiamo su *File*, poi su *New*. Dalla tab *projects* selezioniamo *Series 60 AppWizard*, digitiamo il nome del progetto, settiamo la directory e premiamo *OK*. Nel nuovo dialog inseriamo il nome dell'applicazione e clicchiamo su *Finish*.**

4 **Scriviamo il nostro codice aggiungendo funzioni alla struttura standard o creiamo nuove classi adatte alle nostre esigenze.**

5 **Apriamo una console dei comandi e posizioniamoci nella directory group appartenente al nostro progetto. Digitiamo *bldmake bldfiles* e subito dopo *abld build thumb urel*. Il programma è compilato.**

6 **Per creare il file d'installazione portiamoci invece nella directory *install* e digitiamo *makesis MiaApplicazione .pkg*. Trasferiamo *MiaApplicazione.sis* sul telefonino, installiamolo e lanciamolo.**

```
protocolDescriptorList
->StartListL()
->BuildDESL()
->StartListL()
->BuildUUIDL(KL2CAP)
->EndListL()
->BuildDESL()
->StartListL()
->BuildUUIDL(KRFCOMM)
->BuildUintL(TBuf8<1>(KChannel))
->EndListL()
->EndListL();
CleanupStack::PopAndDestroy(protocolDescriptorList);
```

Ogni funzione della classe *CSdpAttrValueDES* restituisce l'oggetto attivo dopo aver apportato le modifiche. È possibile quindi effettuare una serie di chiamate a cascata. *StartListL* segna l'inizio di una lista di parametri ed *EndListL* ne segna la fine. Tramite *BuildxxxxL* sono stati aggiunti i due protocolli *L2CAP* e *RFCOMM*, discussi in precedenza. Entrambi sono delle costanti definite in *bt_sock.h*. La seguente riga di codice aggiorna il record all'interno del database:

```
sessione.UpdateAttributeL(record,
                    KSdpAttrIdProtocolDescriptorList,
*protocolDescriptorList);
```

Per finire, possiamo aggiornare alcuni singoli attributi del record. Tutte le costanti sono definite in *btsdp.h:*

```
sessione.UpdateAttributeL(iRecord,
    KSdpAttrIdServiceID, KUidBTAdvertiserAppValue);
sessione.UpdateAttributeL(iRecord,
                    KsdpAttrIdBasePrimaryLanguage
    + KSdpAttrIdOffsetServiceName, _L("Mercurio"));
sessione.UpdateAttributeL(iRecord,
                    KSdpAttrIdBasePrimaryLanguage +
                    KSdpAttrIdOffsetServiceDescription,
            _L("Questa è la descrizione del servizio"));
```

Un servizio di nome *Mercurio* è stato aggiunto con successo al database *SDP*. Abbiamo inoltre specificato il suo nome, *Mercurio*, ed una sua descrizione. Aprendo un *ServerSocket* in locale si può adesso sfruttare tale servizio per far conoscere a tutti i dispositivi collegati in rete i parametri necessari per la connessione.

RICERCARE UN SERVIZIO

Come forse molti di noi avranno notato, mandando una foto o un contatto tramite bluetooth, compaiono sul display del nostro cellulare le finestre riportate in **Figura 2**. Questo passaggio è preliminare nel trasferimento delle informazioni in quanto "presenta" due dispositivi, permettendo loro di conoscersi e di dialogare. In questa sezione parleremo di come è possibile rintracciare un servizio. La procedura può essere schematizzata in tre passi:

1. Ricerca e selezione di un dispositivo presente nel raggio d'azione
2. Ricerca dei servizi presenti sul dispositivo, offerti tramite server DSP
3. Selezione di un particolare servizio

RNotifier è la classe chiave per eseguire il primo punto. Dal momento che il suo funzionamento è di tipo server, deve prima essere effettuata una connessione.

```
RNotifier notifier;
User::LeaveIfError(notifier.Connect());
```

Tramite la funzione *StartNotifierAndGetResponse()* è quindi possibile avviare la ricerca dei dispositivi, facendo comparire a schermo una GUI standard che elenca quelli trovati. Notiamo che tale GUI viene visualizzata anche quando il programma non possiede un ambiente grafico. Inoltre, visto che si tratta di una chiamata asincrona , lo stato del procedimento sarà contenuto in un oggetto di tipo *TRequestStatus* e sarà nostra premura attendere su questo oggetto prima di proseguire. Il riferimento al dispositivo selezionato dall'utente verrà memorizzato in un oggetto di classe *TBTDeviceResponseParamsPckg,* che come si vede dalla documentazione Symbian è un *typedef* per il template *TPckgBuf*<>, un contenitore di oggetti di tipo *TBTDeviceResponseParams*. Ecco il codice:

```
TBTDeviceSelectionParamsPckg filtro;
TBTDeviceResponseParamsPckg aResponse;
```

Fig. 2: ***Il dispositivo ricerca i dispositivi e tenta di connettersi a quello selezionato***

NOTA

Il funzionamento server di una classe consente, grazie all'aiuto dei thread, l'accesso contemporaneo di diversi processi. Questo risulta utile quando le azioni che vogliamo implementare sono abbastanza comuni e possono essere eseguite nello stesso momento.

```
TRequestStatus status;
notifier.StartNotifierAndGetResponse( status,
        KDeviceSelectionNotifierUid, filtro, aResponse );
User::WaitForRequest(status);
if (status.Int() == KErrNone)
        User::Panic(_L("RicercaDispositivi"),-1);
notifier.CancelNotifier(KDeviceSelectionNotifierUid);
notifier.Close();
```

È adesso possibile ottenere tutti i servizi disponibili relativi all'indirizzo del device selezionato in precedenza dall'utente. Il sistema operativo ci mette a disposizione un agente tramite la classe *CSdpAgent*. I parametri di ingresso nella costruzione sono due: un oggetto che implementa l'interfaccia *MSdpAgentNotifier* ed un indirizzo.

```
CSdpAgent* agente = CSdpAgent::NewL(*this,
                        aResponse.BDAddr());
agente->NextRecordRequestL();
```

NextRecordRequestL() fa partire la scansione. Ogni qual volta viene individuato un servizio, l'agente richiama *NextRecordRequestComplete()* sull'oggetto passato in fase di costruzione. Nel nostro caso è *this*. Ecco la segnatura della funzione:

```
void NextRecordRequestComplete(TInt aError,
                TSdpServRecordHandle aHandle,
                TInt aTotalRecordsCount)
```

All'interno di tale funzione dobbiamo scrivere il codice per verificare se si tratta del servizio che stiamo cercando. Tutte le informazioni sui servizi sono ovviamente contenute nel parametro *aHandle*.

COLLEGAMENTO POINT TO POINT

Avendo visto come pubblicare un servizio e come cercarne di nuovi, possiamo addentrarci in un caso pratico. L'SDK della Nokia contiene, tra gli applicativi d'esempio, proprio quello che fa al caso nostro. Il ricevente crea un ServerSocket su una data porta. I parametri di connessione vengono poi incapsulati in un servizio che viene aggiunto al database SDP. Il dispositivo mandante effettua quindi la ricerca di tale servizio tramite la procedura riportata nella sezione precedente. Una volta stabilito il collegamento, i due processi, residenti su due diversi dispositivi, possono trasferirsi informazioni. Senza scendere troppo nello specifico diciamo che le classi interessate sono *RSocketServ* e *RSocket*. Esse contengono le classiche funzioni attuabili sui socket, come *open()*, *bind()*, etc. Basta dare uno sguardo alla documentazione Symbian per creare in pochi passi un applicativo point to point.

Un'ultima constatazione deve riguardare la relazione tra i socket e i servizi bluetooth.

I servizi servono proprio per far circolare sulla rete le informazioni riguardanti il collegamento tra due socket, primi fra tutti l'indirizzo di rete e la porta di connessione.

In più, dato che si tratta di una tecnologia molto versatile, le informazioni trasportabili col servizio stesso possono variare in numero e dimensione.

COS'È UNA CHIAMATA ASINCRONA

Una chiamata asincrona viene eseguita in maniera parallela al normale flusso di codice. A seconda dei casi, è necessario aspettare esplicitamente che essa termini, prima di poter effettuare altre operazioni. Una chiamata sincrona è anche definita bloccante in quanto il programma non continua la sua esecuzione fintantoché la funzione sta lavorando.

Fig. 3: ***Sul sito di Nokia www.forum.nokia.com/main/0,,034-4,00.html è possibile trovare un elevato numero di informazioni sulla programmazione Symbian***

CONCLUSIONI

Siamo adesso in grado di aggiungere un seppure minimo supporto bluetooth all'interno delle nostre applicazioni. Il discorso sarebbe molto più vasto.

Si potrebbe parlare di OBEX, un protocollo per il trasferimento di file in una rete senza fili. Oppure analizzare la sicurezza in una rete bluetooth. Symbian stesso dispone di un certo numero di API per l'autenticazione e per la crittazione delle informazioni. Come sempre, una volta fatte le fondamenta, la costruzione è molto più facile. Infine, a noi piace molto quella meravigliosa sfida che è la programmazione.

Antonio Trapani

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

VISUAL BASIC

INTERCETTARE GLI EVENTI DEL CD-ROM

In taluni casi può risultare utile intercettare l'inserimento e l'estrazione di un cd-rom o di un altro media rimovibile nel computer, per proseguire con ulteriori indagini sul volume inserito o semplicemente per aggiornare alcuni elementi dell'interfaccia grafica. In Windows è possibile ricevere le notifiche di tali eventi semplicemente gestendo il messaggio WM_DEVICE_CHANGE. Questo tip illustra come fare a ridefinire il metodo WndProc di un'applicazione windows form per gestire tale messaggio.

```
'Costanti usate dal gestore dei messaggi:
Private Const WM_DEVICECHANGE As Integer = 537
Private Const DBT_DEVICEREMOVECOMPLETE As Integer = 32772
Private Const DBT_DEVICEARRIVAL As Integer = 32768
Protected Overrides Sub WndProc(ByRef m As
                                    System.Windows.Forms.Message)
    Select Case m.Msg
        Case WM_DEVICECHANGE
            If (m.WParam.ToInt32() = DBT_DEVICEREMOVECOMPLETE) Then
                MessageBox.Show("CD espulso!")
            ElseIf (m.WParam.ToInt32() = DBT_DEVICEARRIVAL) Then
                MessageBox.Show("Nuovo CD inserito!")
            End If
        Case Else
            'Chiama l'implementazione base di WndProc per gli altri
                                                        messaggi:
            MyBase.WndProc(m)
    End Select
End Sub
```

JAVA SCRIPT

AGGIORNARE SENZA REFRESH DEL BROWSER

Con questo semplice esempio mostriamo come sia possibile, utilizzando l'oggetto XMLHttpRequest, realizzare delle pagine che aggiornano la loro interfaccia senza richiedere il refresh del browser. Lo script che segue rende possibile realizzare una pagina che mostra una serie di slide, prelevate su richiesta dal server, e le visualizza senza ricaricare la pagina. Per provare l'esempio è necessario pubblicare su un server web lo script, la pagina di prova e le slides.

```
/**
 * Funzione che istanzia un oggetto XMLHttpRequest usando un
                                    meccanismo cross browser.
 *
 * @return   restituisce un'istanza di XMLHttpRequest oppure il
                                         valore false in caso
 *           di errori.
 */
function getXMLHttpRequestInstance()
{
    var xmlhttp;
    // Prova il metodo Microsoft usando la versione più recente:
    try
    {
        xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e)
    {
        try
        {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (E)
        {
            xmlhttp = false; } }
    // Se non è stato possibile istanziare l'oggetto forse siamo
    // su Mozilla/FireFox o su un altro browser compatibile:
    if (!xmlhttp && typeof XMLHttpRequest != 'undefined')
    {
        try
        {
            xmlhttp = new XMLHttpRequest();
        } catch (e)
        {
            xmlhttp = false; } }
    // Restituisce infine l'oggetto:
    return xmlhttp; }
/**
```

```
 * Funzione che sostituisce il contenuto HTML di un nodo della pagina.
 *
 * @param    nodeId ID del nodo
 * @param    html   codice HTML da sostituire a quello del nodo
 */
function updateContent(nodeId, html)
{
    var node = document.getElementById(nodeId);
    if(null == node)
    {
       alert("[ERRORE] L'elemento " + nodeId + " non esiste");
       return; }
    node.innerHTML = html;
    node.style.visibility = "visible";}
/**
 * Richiede al web server il contenuto di una slide (testo o HTML) in
                                                  maniera asincrona.
 * @param nodeId ID dell'elemento della pagina che conterrà la slide
 * @param url URL della slide (deve essere sullo stesso server per
                                                  motivi di sicurezza)
 */
function showSlide(nodeId, url)
{
   var xmlhttp = getXMLHttpRequestInstance();
   if(!xmlhttp)
   {
      alert("Il browser non supporta l'oggetto XMLHttpRequest");
      return false;}
    xmlhttp.open("GET", url,true);
    xmlhttp.onreadystatechange=function()
    {
        if (xmlhttp.readyState==4)
        {
            if (xmlhttp.status==200)
            {
                updateContent(nodeId, xmlhttp.responseText);
            } else if (xmlhttp.status==404) {
                alert("[ERRORE] l'URL "+url+"non esiste!");
            } else {
                alert("[ERRORE] errore non gestito (" + xmlhttp.status
                                                          + ")");}
   } }
   xmlhttp.send(null);
}
```

IL TIP DEL MESE

AGGIUNGERE EFFETTI SPECIALI ALLE FINESTRE

Tip fornito da Domenico Testa

Diamo un pò di brio alle nostre applicazioni Windows Forms aggiungendo qualche gradevole effetto speciale. Possiamo utilizzare infatti, tramite i servizi di interoperabilità messi a disposizione dal .NET framework, la funzione AnimateWindow, esportata dalla libreria di sistema user32.dll.

Con questa funzione è possibile, specificato l'handle di una finestra, aggiungere e combinare tra loro una serie di effetti speciali quali dissolvenza, scorrimento orizzontale e verticale, sia al caricamento della finestra che alla sua chiusura, semplicemente passando gli opportuni flag.

```
using System;
using System.Windows.Forms;
class AnimatedForm : Form
{  /// <summary>
   /// Funzione per animare una finestra, esportata dalla libreria
                                                            user32.dll
   /// </summary>
   [System.Runtime.InteropServices.DllImport("user32.dll")]
   static extern bool AnimateWindow(IntPtr hwnd, int time,
                                          AnimateWindowFlags flags);
   /// <summary>
   /// Flag utilizzabili nella funziona AnimateWindow.
   /// Questi flag possono essere combinati.
   /// </summary>
   enum AnimateWindowFlags
   { HorPositive= 0x00000001, // Anima la finestra da sx a dx (con
                                                      AnimationSlide)
      HorNegative = 0x00000002, // Anima la finestra da dx a sx
                                                  (con AnimationSlide)
      VerPositive = 0x00000004, // Anima la finestra dall'alto verso
                                                              il basso
      VerNegative = 0x00000008, // Anima la finestra dal basso
                                                             vero l'alto
      Center = 0x00000010, // Centra la finestra
      Hide = 0x00010000, // Nasconde la finestra
      Activate = 0x00020000, // Attiva la finestra
      Slide = 0x00040000, // Usa l'animazione slide
      Blend = 0x00080000  // Attiva un effetto fading }
   AnimatedForm()
  { // Qualche effetto speciale d'esempio:
     AnimateWindow(Handle, 500, AnimateWindowFlags.Activate |
                                            AnimateWindowFlags.Blend);
     AnimateWindow(Handle, 700, AnimateWindowFlags.Hide |
          AnimateWindowFlags.Slide | AnimateWindowFlags.Center);
     AnimateWindow(Handle, 500, AnimateWindowFlags.Slide |
                                       AnimateWindowFlags.HorPositive);
     AnimateWindow(Handle, 500, AnimateWindowFlags.Hide |
       AnimateWindowFlags.Slide | AnimateWindowFlags.HorNegative
                                     | AnimateWindowFlags.VerPositive);
    AnimateWindow(Handle, 500, AnimateWindowFlags.Activate |
      AnimateWindowFlags.Slide | AnimateWindowFlags.HorPositive |
                                  AnimateWindowFlags.VerNegative); }
   static void Main()
   {  Application.Run(new AnimatedForm()); }}
```

Input guidato in dB Access

Spesso la fase di inserimento dei dati può produrre errori. I principi sull'usabilità del software indicano chiaramente la strada che il programmatore deve seguire. L'input dei dati deve essere semplice e deve ridurre al minimo gli errori. Esistono molti accorgimenti che vanno in questa direzione. Se il valore da inserire è contenuto in un insieme circoscritto, ossia un gruppo di elementi non molto numeroso; allora si può pensare di proporre all'utente la scelta tra una lista. Digitare il valore da immettere potrebbe provocare un errore di battitura. La scelta tra una lista di valori minimizza la possibilità di sbaglio. Nell'esempio si fa riferimento ad un database di nome software costruito con MS Access. Esiste un'unica tabella di nome software che ha lo scopo di mantenere informazioni circa software commerciali, una sorta di biblioteca di programmi. Se volete una " softwareteca". Per ogni record si memorizza un IDS – identificativo del software che è anche chiave. Gli altri attributi sono: nome, casap, versione, tipo e licenza. I primi tre indicano rispettivamente la denominazione, la casa produttrice e la release del software. Il loro input avviene in modo tradizionale, per completa digitazione. I successivi due sono il tipo (di base, applicativo, e altri) e la licenza (freeware, shareware e altro). Come suggeriscono le parentesi, per questi, si possono costruire insiemi ridotti di elementi.

Fabio Grimaldi

‹1› GENERA LA MASCHERA IN MODALITÀ STRUTTURA

Dopo aver eseguito le operazioni di routine: apertura ambiente Access; creazione di un database vuoto; salvataggio con nome software; creazione di una tabella con i campi sopra descritti, si genera una maschera in modalità *struttura.* Si associa alla tabella software. Apparirà la tabella, sottoforma dei sui campi di fianco alla maschera.

‹2› FASE DI COSTRUZIONE DELLA MASCHERA

Trascinando i campi di cui si vuole digitare l'input in automatico si creano delle caselle di testo. Utilizzando la casella degli strumenti si aggiungono due caselle combinate. Associate rispettivamente ai due campi tipo e licenza. Si personalizza la form con un titolo ed eventuali immagini. L'aspetto ottenuto è quello in figura.

‹3› IMMISSIONE PERSONALIZZATA DI UNA LISTA DI VALORI

Se la casella degli strumenti ha attivato il generatore automatico di eventi, viene avviato un processo per l'immissione dei valori. A tale scopo il bottone con la bacchetta magica deve essere in stato di premuto. Noi optiamo per l'immissione personalizzata. Per poter aggiungere una lista di valori da cui l'utente potrà attingere. Questo ci darà maggiore flessibilità.

‹4› IL CAMPO TIPO È A DUE COLONNE

Col1	Col2
Di base	Sistema Operativo
~	Utility
~	Antivirus
~	Gestione rete
Applicativo	Office Automation
~	Programmazione
~	DataBase

Per rendere l'input più professionale per il campo tipo associamo due colonne. Per potere distinguere i due grandi gruppi di software di base e applicativo. Si digita *2* al numero di colonne. Si tratta però di specificare che sarà la seconda colonna ad andare nel campo tipo. Questo lo si decide al passo successivo.

‹5› ASSOCIAZIONE AL GIUSTO CAMPO

Nella finestra successiva si associa il risultato al campo tipo. Si ripete dal passo due la stessa sequenza di operazioni per il campo licenza. Questa volta è sufficiente una sola colonna. Adesso l'intera maschera è completa. Usciamo e diamo un nome alla form. Input software è un etichetta appropriata.

Non resta che vedere il risultato.

‹6› OBIETTIVO RAGGIUNTO. ECCO LA MASCHERA IN FUNZIONE

Per vedere il risultato basta cliccare sulla maschera appena creata. Si nota come i primi tre input debbano essere digitati, mentre i secondi due possano essere effettuati mediante la scelta di un valore su una casella combinata.

La chiave IDS essendo contatore viene assegnata in automatico mediante incremento.

Query a campi incrociati

Estrarre informazioni dai dati opportunamente organizzati in un database è uno dei compiti di un avanzato DBMS. In MS Access, tra gli altri strumenti, sono previste le query a campi incrociati. Si tratta di un'utile mezzo che permette di riorganizzare e computare i dati per semplificarne l'analisi. Le query a campi incrociati vengono utilizzate per eseguire una somma, una media, un conteggio o qualsiasi altro tipo di totale sui dati raggruppati in base a due tipi di informazioni. Tali informazioni saranno disposti su una tabella a doppia entrata, ovvero, una tabella in cui le intestazioni di riga e di colonna sono associate alle due informazioni. L'incrocio tra riga e colonna è l'elemento che attraverso le nostre query intendiamo riorganizzare. Nell'esempio si intende applicare il metodo ad una tabella ottenuta come il risultato di una query. Essa conteneva i dati delle spese in relazione a dei progetti ed in particolare alle voci di tali progetti. Le voci sono le stesse per differenti progetti. La classica visualizzazione della tabella, produceva innanzitutto ridondanza poiché le stesse voci venivano ripetute per diversi progetti, così come si duplicava il nome del progetto; inoltre, la lettura era poco chiara. Applicando il metodo citato elimineremo la ridondanza e otterremo una tabella leggibile e significativa. Vediamo come.

Fabio Grimaldi

‹1› SITUAZIONE DI PARTENZA

PROGETTI : Tabella

Progetto	Voce	Spesa
Editoria elettronica	Progettazione	2300
Editoria elettronica	Materiale	5100
Editoria elettronica	Consulenza	10800
L'azienda nel web	Progettazione	1800
L'azienda nel web	Materiale	3100
L'azienda nel web	Consulenza	12500
		0

Inizialmente si ha una tabella in cui sono presenti tre attributi. I primi due identificano rispettivamente il progetto e la voce di spesa. Nell'ultimo attributo è riportata la spesa vera e propria. Si può notare presenza di ridondanza e la difficoltà nel leggere i dati, che in una sola parola sono disorganizzati. Il nostro scopo è ottenere una tabella che riassuma le voci di spesa per ogni tipologia di dato; ad esempio ci piacerebbe sapere quanto si spende globalmente.

‹2› PRODUCIAMO UNA QUERY

Anche se si tratta di riorganizzare i dati e non di selezionarli, ci serviamo ugualmente di una query. È questa l'occasione per ampliare il significato di questo utile strumento. Per farlo scegliamo dal raccoglitore apposito, che compare all'avvio di Access, lo strumento query che applicheremo alla tabella progetti già presente.

‹3› OPTIAMO PER LA QUERY A CAMPI INCROCIATI

Una volta avviata la procedura di strutturazione della query si procede con la scelta del tipo. Dalla barra dei menu si può operare tale scelta. Nel caso specifico optiamo per la query a campi incrociati selezionando dal sotto menu query la voce apposita. Al termine dell'operazione apparirà la ben conosciuta finestra di produzione query.

‹4› STRUTTURAZIONE DELLA QUERY

Campo:	Progetto	Voce	Spesa
Tabella:	PROGETTI	PROGETTI	PROGETTI
Formula:	Raggruppamento	Raggruppamento	Somma
Campi incrociati:	Intestazione riga	Intestaz. colonna	Valore
Ordinamento:			
Criteri:			
Oppure:			

I due campi progetto e voce saranno rispettivamente le intestazioni della riga e della colonna.
Per farlo si deve porre a raggruppamento il valore corrispondente alla riga formula, e intestazione di riga (di colonna per l'attributo voce) in corrispondenza di campi incrociati.
Per spese i due campi varranno somma e valore.

‹5› METTERE APPUNTO E SALVARE LA QUERY

Nella modalità struttura i valori previsti dalla query a campi incrociati possono essere selezionati da apposite liste. Appare un piccolo menu accessibile attraverso la conosciuta freccetta verso il basso. Terminato la definizione della struttura della query, chiudendola ci viene chiesto il nome con cui salvarla.
Scegliamo *"Spese X progetto e X voce"*.

‹6› VISUALIZZAZIONE DELLA QUERY A CAMPO INCROCIATO

Spese X progetto e X voce : Query a campi incrociati

Progetto	Consulenza	Materiale	Progettazione
Editoria elettronica	10800	5100	2300
L'azienda nel web	12500	3100	1800

Il risultato è davvero apprezzabile. I valori di spesa sono incasellati con un tabella che ricorda un po' le griglie di Excel. Del resto anche access deve prevedere il trattamento, se pur in forma più elementare di dati numerici. Si può notare la maggiore chiarezza e l'eliminazione della ridondanza rispetto alla situazione di partenza di *Figura 1*.

Realizzare una classe per la gestione delle matrici in c++

Spesso capita di dover lavorare con dati ordinati come tabelle oppure con le matrici. In questi casi ci troviamo di fronte a due problematiche: la gestione dinamica della memoria e l'accesso rapido alla struttura dati. La maggior parte dei casi permette l'utilizzo delle matrici statiche. Ovvero vengono fissati il numero di righe ed il numero di colonne al momento della creazione e non abbiamo più la possibilità di modificarle.
Ma non di rado incontriamo il bisogno di dover cambiare la dimensione ed allora incontriamo dei problemi. Quindi ora andiamo a vedere come realizzare una classe template Matrice veloce e semplice da usare. Ciò di cui abbiamo bisogno è un compilatore c++, un editor di testo e un po' di dimestichezza con c++.
Per realizzare il seguente esempio ho utilizzato Dev-C++ un ambiente di sviluppo c++ freeware.
Dev-C++ è scaricabile gratuitamente dal sito web: *www.bloodshed.net*

Stefano Vena

‹1› LE PRIME RIGHE DI CODICE

```
#include <stdlib.h>
template<class T>
class Matrice
{
  private:
   T* vettore;
   int size;
   int m_cols;
   int m_rows;
```

La classe viene realizzata come template in modo tale da rendere semplice l'utilizzo della stessa per qualsiasi tipo di dato. La "specializzazione" per un tipo di dato verrà fatta dal compilatore durante la compilazione. I membri della classe vengono definiti con visibilità private. Da notare la variabile che conterrà la matrice è un semplice vettore!

‹2› I COSTRUTTORI

```
public:
  Matrix()
  {
    vettore = NULL;
    size = 0;
    m_cols = m_rows = 0; }
  Matrix(int rows, int cols)
  {
    vettore = NULL;
    resize(rows,cols);
    reset(); }
  Matrix(const Matrix<T>& m)
  {
     vettore = NULL;
     resize(m.m_rows,m.m_cols);
     memcpy(vettore,m.vettore,size); }
 ~Matrix()
 {
    if(vettore!=NULL)free(vettore); }
```

Trattandosi di una classe template dobbiamo implementare i metodi in modalità inline poiché non tutti i compilatori supportano le classi template nel formato *"Intestazione – Sorgente"*. Il codice inserito in questo passo non necessita altre spiegazioni.

‹3› ALLOCAZIONE DELLA MEMORIA

```
void resize (int rows, int cols) {
  size = sizeof(T) * rows * cols;
if(vettore == NULL)
  vettore = (T*) malloc ( size);
else
 vettore = (T*) realloc (vettore, size);
if(vettore == 0)
       throw "Memoria insufficiente";
   m_cols = cols;
   m_rows = rows;
 }
```

La funzione resize è il cuore della classe. Attraverso questo metodo ridimensioniamo la matrice.
La variabile vettore è un puntatore, quindi se esso punta al valore *NULL* allochiamo lo spazio necessario per la prima volta con *malloc.* Se abbiamo già allocato della memoria per il puntatore vettore ne riserviamo altra, secondo le nostre esigenze tramite la funzione *realloc.* Se la memoria virtuale non è sufficiente a soddisfare la richiesta, la nostra variabile punterà al valore *NULL*. In questa sfortunata situazione lanciamo un eccezione. Sempre in questa fase salviamo le informazioni su righe e colonne della matrice.

‹4› ACCESSO ALLE CELLE

```
const T& getAt(int row,int col) const {
 return vettore[row*cols+col];}
void setAt(int row,int col,
const T& value){
   vettore[row*cols+col] = value;}
const T*  operator[](int row) const{
return vettore+row*m_cols;}
T* operator[](int row){
 return vettore+row*m_cols;
}
```

Il metodo di memorizzazione delle celle è molto semplice ed efficiente.Le righe vengono salvate di seguito una all'altra. Per ottenere il valore alla cella di coordinate *x,y* "saltiamo" le prime *x* righe e restituiamo il valore alla colonna *y*. I metodi *getAt* e *setAt* sono molto intuitivi e non necessitano di altri commenti. L'operatore *[]* è realizzato in modo tale da simulare il comportameto dell'operatore *[] []*. Infatti utilizzando l'operatore *[]* esso restituisce un sotto array che punta alla prima cella della riga scelta.
Accederemo alle celle successive attraverso l'utilizzo della doppia parentesi quadra(*[]*) ancora una volta.

‹5› ALTRE OPERAZIONI

```
int cols ()const {return m_cols;}
int rows ()const {return m_rows;}
void reset()
{
 if( vettore == NULL ) return;
 memset(vettore,0,size);
}
};
```

A questo punto aggiungiamo la possibilità di ottenere il numero di righe e colonne della matrice.
Se vogliamo impostare tutte le celle della matrice sul valore zero possiamo usare il metodo *reset*.
Grazie alla funzione memset il metodo reset azzera la matrice.

‹6› UTILIZZIAMO LA CLASSE

```
Matrice<int> Matrice_Int (,5);
Matrice<double> Matrice_Dbl;

Matrice_Dbl.resize(10,5);
Matrice_Dbl[0][0] = 1.0;
Matrice_Dbl.setAt(1,1 ,
           Matrice_Dbl.getAt(0,0) );

cout << Matrice_Int.cols() << endl;
cout << Matrice_Int.rows() << endl;

Matrice_Int.resize();
```

Questa è una carrellata delle poche ma utili funzionalità della classe appena creata.

Realizzare una classe per la gestione "Intelligente" dei puntatori

I puntatori rappresentano gioie e dolori dei programmatori C++. Essi permettono di compiere tutte quelle operazioni che fanno parte della programmazione "sporca" ma proprio per questo motivo sono spesso la causa di errori. Capita spesso di utilizzare puntatori che non fanno più riferimento agli oggetti aspettati, oppure tentiamo di liberare la memoria di puntatori vuoti o peggio ancora terminiamo il programma senza rilasciare le risorse. In questi casi avremmo bisogno del garbage collector! In mancanza del garbage collector possiamo realizzare e, quindi, utilizzare una classe in grado di maneggiare i puntatori al posto nostro. Attenzione però! Gli oggetti della nostra classe andranno istanziati tutti sullo stack e non nella memoria virtuale! Quindi ora andiamo a vedere come realizzare un gestore di puntatori attraverso una classe template veloce e semplice da usare. Ciò di cui abbiamo bisogno come al solito è un compilatore c++, un editor di testo e un po' di dimestichezza con il linguaggio. Per realizzare il seguente esempio potremmo utilizzare l'ambiente di sviluppo (freeware). Dev-C++, scaricabile gratuitamente dal sito web : *www.bloodshed.net*

Stefano Vena

‹1› LE PRIME RIGHE DI CODICE

```
#ifndef PUNTATORI_INT_H
#define PUNTATORI_INT_H
#include <map>
extern
std::map<void *,int> lista_puntatori;
template <class T>
class Puntatore
{  protected:
      T *obj;
```

Per prima cosa creiamo due file il primo di nome *"puntatore.h"* ed un secondo di nome "puntatore .cpp" nel file con estensione *"h"* andiamo a scrivere il codice del primo passo. Gioca un ruolo importante la mappa *"lista_puntatori"* in essa vengono salvati gli indirizzi di tutti i puntatori utilizzati. Essa viene dichiarata all'esterno della classe in modo tale da essere condivisa da tutti gli oggetti di tipo *"Puntatore"*. La classe in questione è un template e contiene un solo membro interno che rappresenta il puntatore da gestire.

‹2› I COSTRUTTORI

```
public:
Puntatore()
{
  obj = NULL;
}

Puntatore(T * _obj)
{
  obj = NULL;
  Assegna(_obj);
}

~Puntatore()
{
  Assegna(NULL);
}
```

Trattandosi di una classe template dobbiamo implementare i metodi in modalità inline poiché non tutti i compilatori supportano le classi template nel formato *"Intestazione – Sorgente"*.
I costruttori sono solo due uno è quello di default e fa puntare il puntatore gestito a *NULL*.
L'altro prende come parametro un puntatore ad un oggetto di tipo *T* e lo assegna al gestore.
Il codice inserito in questo passo non necessita altre spiegazioni.

‹3› ASSEGNA E RILASCIA

```
const Puntatore &Assegna(T *_obj)
{  T *oldobj=Rilascia();
  if (oldobj)
  { if (oldobj!=_obj) delete oldobj; }
  else
  { obj = _obj; }
  if (obj)
    lista_puntatori[(void *)obj]++;
  return *this; }
T *Rilascia()
{  T *oldobj = obj;
    if (obj)
   { if((lista_puntatori[(void *)obj]--)<=0)
     { lista_puntatori.erase((void *)obj); } }
  obj = NULL;
  return oldobj; }
```

Le funzioni *Assegna* e *Rilascia* rappresentano il nucleo della classe. Attraverso la funzione *"Assegna"* possiamo creare un nuovo oggetto oppure determinare un nuovo utilizzo. Ogni qual volta invochiamo il metodo *Assegna* otteniamo il puntatore all'oggetto corrente e se l'oggetto è nuovo cancelliamo il vecchio. Dopo questa operazione incrementiamo il contatore Su gli accessi all'oggetto corrente. Attraverso la funzione *"Rilascia"* possiamo rilasciare il puntatore correntemente gestito e decrementare il contatore sugli accessi, quindi se il valore del contatore scende sotto lo zero lo eliminiamo dalla mappa.

‹4› OPERATORI

```
T *get(){ return obj; }
T &operator*()
{ if (!obj) Assegna( new T());
  return *get(); }
T *operator->()
{  if (!obj) Assegna( new T() );
   return get(); }
const Puntatore &operator =(T *_obj)
{ return Assegna(_obj); }
operator T*()
{ return get(); } };
#endif
```

L'overloading di questi operatori ci permette di utilizzare le istanze degli oggetti di tipo *Puntatore* come dei comuni puntatori agli oggetti con la differenza che quando tutti i riferimenti al puntatore gestito verranno meno questo verrà eliminato.

‹5› IL FILE PUNTATORE.CPP

```
#include "puntatore.h"
#include <stdio.h>
std::map<void *,int> lista_puntatori;
```

A questo punto andiamo ad aggiungere le ultime righe di codice al file *"puntatore.cpp "*.
Avendo già implementato i metodi della classe all'interno dell'intestazione ora non ci resta che dichiarare la mappa.
Fatto questo siamo pronti per utilizzare i nostri puntatori intelligenti.

‹6› UTILIZZIAMO LA CLASSE

```
Puntatore<Obj> obj = new Obj(1);
Puntatore<Obj> obj1 = obj;
cout << obj->Metodo() << endl;
cout << obj1->Metodo() << endl;
cout << (*obj).Metodo() << endl;
cout << ( (Obj*) obj )->Metodo() << endl;
```

Questa è una carrellata delle potenti e utili funzionalità della classe appena creata.

Esportazione di un modello di progetto C++ in HTML

Siamo abituati a trattare con linguaggi e programmi che siano dotati di estensioni in grado di farli comunicare con altri linguaggi e programmi. Dev C++ è uno di essi. Il C++ free di bloodshed, infatti, è in grado di esportare le proprie produzioni in diversi formati in modo potente e naturale. I due formati previsti sono i più utili e comuni: il famoso *Rich text Format –RTF-* come formattatore di testi e *Hyper text mark up language – HTML* - il noto linguaggio di marcatura per il web. Nell'esempio attuale siamo interessati a questo secondo formato. Vedremo come con pochi colpi di click sia facilmente ottenibile una pagina web, appunto in formato HTML, che contiene le informazioni riassuntive sul codice sviluppato. Un utilissimo strumento per chi, oltre a produrre codice, ha la necessità di pubblicare costantemente in rete il proprio lavoro; o comunque di mantenere ordinato il proprio archivio di codice sviluppato organizzandolo come un ipertesto. In particolare, vedremo come sia possibile tradurre in formato HTML un progetto C++. Per farlo apriremo un semplice progetto reperibile negli esempi di cui è corredato Dev C++. Il semplice programma che visualizza a video una stringa. Lanciamo quindi l'applicazione e dall'amichevole ambiente di sviluppo selezioniamo dal menu file apri progetto.

Fabio Grimaldi

<1> PER COMINCIARE APRIAMO IL PROGETTO

Il progetto di esempio a cui facciamo riferimento è Wintest, che contiene il file test.c. Esso è reperibile nella directory di esempi allegata all'applicativo Dev-C++. Ovviamente, per aprirlo basterà selezionare dal menu file la voce apri progetto e individuare la cartella di esempio, quindi il progetto nominato.

<2> COMPILAZIONE, ESECUZIONE E TEST

Vediamo di cosa si tratta. È il più semplice dei programmi che si possa costruire. Hello word ciò che si tenta di visualizzare la prima volta che si esplora un programma. Cliccando sull'apposito bottone, o selezionando l'analoga voce dalla barra dei menu si compila ed esegue l'intero progetto. Successivamente, si può verificare il l'applicazione in funzione.

<3> ESPORTAZIONE DELL'INTERO PROGETTO

Dal menu file optiamo per la voce esporta, e dal sotto menu che compare scegliamo progetto in HTML.
È questa la funzione che in automatico svolge la funzione che abbiamo descritto.
Adesso si tratta soltanto di specificare dove e con quale nome salvare il progetto esportato in formato HTML.

<4> SALVATAGGIO DEL PROGETTO IN HTML

La tipica finestra di dialogo windows ci consente di scegliere percorso e nome del file HTML.
Diamo alla pagina web in fase di creazione il nome *prog* il file HTML verrà salvato nella derectory selezionata.
Basterà essere muniti di un qualsiasi browser per poter richiamare e visualizzare il risultato ottenuto.

<5> VISIONE DEL RISULTATO. ECCO LA PAGINA HTML

Da sistema operativo, accedendo al percorso del file e cliccando su di esso, si può verificare il risultato ottenuto. Come si può notare, si tratta di una semplice quanto utile pagina che riassume le informazioni salienti del progetto.
Sono presenti, inoltre, due link: al software che ha prodotto il codice e al codice stesso.

<6> DIAMO UN'OCCHIATA AL CODICE DI ESEMPIO

Per completare il percorso esplorativo verso questa funzionalità di DEV-C++, clicchiamo sul nome del file e visioniamo il codice.
Anch'esso è riformattato in HTML. Pronto a essere copiato e incollato da altri utenti che lo consultano ad esempio su internet. Una riprova della completezza informativa che tale esportazione fornisce.
Si tratta di una soluzione utile per creare ad esempio documentazione relmativa a un progetto a cui si sta lavorando o per rendere disponibile sul web o su un cdrom il listato dei nostri progetti.
Anche in questo DevC++ si dimostra un editor solido completo di tutte le funzionalità anche importanti che spesso si ritrovano in IDE commerciali.

Sun Java Studio Enterprise

traduzione a cura di Milena Ianigro

Abbiamo intervistato Chris Atwood il team manager di Sun Java Studio, a proposito delle novità del nuovo ambiente e più in generale su come vede lo sviluppo di Java Ecco cosa ci ha detto

ioProgrammo: Prima di tutto complimenti per l'ottimo lavoro che Sun sta svolgendo. I progressi di Java e dei tools che gli ruotano intorno sono davvero notevoli. Tanto per rompere il ghiaccio, ci parli un po' di Java Studio Enterprise. Che cosa è? A chi è diretto?
Chris Atwood: L'obiettivo di Sun Java Studio Enterprise è di aumentare la produttività dei progettisti e dei team di sviluppo. Il miglioramento della produttività risulta evidente nel ciclo di vita proposto dall'applicativo, che include la documentazione e modellazione del progetto, la collaborazione fra team dislocati in diversi Paesi, lo sviluppo e il debugging rapido dei servizi di sistema Java e infine un miglior controllo delle prestazioni una volta effettuato il deploy.

Chris Atwood

ioP: Il cuore dell'intero Java Studio Enterprise sembra essere il "Code Aware Collaboration", cioè il sistema che consente a team distribuiti di lavorare in collaborazione: è così?
C.A.: La collaborazione istantanea fra programmatori è una direzione alla quale credo che lo sviluppo debba tendere. Bisogna orientarsi verso progetti distribuiti, all'interno dei quali gli analisti lavorano a stretto contatto con i clienti e direzionano i loro sforzi concordandoli con il team degli sviluppatori. La presence-awareness (la possibilità di sapere se i nostri interlocutori sono online) implementata in Java Studio rende più rapidi tali sforzi di revisione di codice e di specifica, specialmente se unita al servizio di collaborazione Java System ora disponibile su java .net. Oltre a queste nuove funzionalità orientate al lavoro in team, Java Studio Enterprise include strumenti quali refactoring, portlet builder, app server, database, portal server, access management, e directory server. Infine senza nessun costo aggiuntivo è possibile accedere al supporto per gli sviluppatori tramite i forum e le email gestite dal personale di Sun.

ioP: Abbiamo trovato molto interessante l'integrazione con UML, anche se UML non è ancora molto diffuso fra gli sviluppatori meno esperti.
Pensa che Java Studio possa colmare questo gap?
C.A.: Abbiamo creato Java Studio per fornire agli analisti professionisti uno strumento in grado di velocizzare le proprie esigenze di design e di implementazione. E comunque forniamo come parte integrante di Java Studio tutorial online, video e webchat che possono essere utilizzati dai programmatori per migliorare la propria conoscenza di UML. Inoltre, sia i programmatori esperti che quelli meno esperti dovrebbero informarsi sui Sun Developer Tech Days durante i quali teniamo corsi di formazione e conferenze nelle varie città.

ioP: A suo tempo Microsoft fu accusata di aver plagiato Java con C# e la piattaforma .NET. Ora invece sembra che sia Sun a rincorrere la facilità d'uso e la semplicità tipica di Microsoft. Qual è la sua opinione?
C.A.: Innanzitutto devo dire che questo è un ottimo momento per noi programmatori, tutti si danno un gran da fare per avere la nostra attenzione! E questa gara all'accaparramento delle menti ha portato molti sviluppatori a scegliere di stare in mezzo: 4.5 milioni di sviluppatori (e la cifra è in crescita) si sono orientati verso la compatibilità e la sicurezza offerti da Java, caratteristiche che nei 10 anni passati sono state arricchite con un notevole insieme di API. Grazie a questa vasta base di sviluppatori Java già installata, un gran numero di partner e di community stanno collaborando a strumenti che semplificano in modo considerevole la creazione di nuovi servizi pur preservando la compatibilità dello standard Java. Un esempio di questa semplificazione in evoluzione è l'engineering bidirezionale che è presente sia in Java Studio Enterprise sia in Creator. In Creator l'applicazione è disegnata su un canvas visivo e il codice standard di Java è generato a partire da questo modello. Allo stesso modo, quando il codice viene modifi-

cato anche la parte visuale viene modificata di conseguenza. Non potrebbe essere più semplice. Lo stesso dicasi per i progettisti che usano Java Studio Enterprise: il diagramma delle classi UML è generato a partire dal codice Java e quando il modello viene modificato si genera anche il nuovo codice. Ancora una volta i vantaggi della piattaforma Java sono da ricercare nella creazione di servizi fruibili dai clienti indipendentemente dalla piattaforma che utilizzano. Al di là della facilità e della semplicità d'uso degli strumenti, Java possiede una semantica più ampia, che ha consentito agli sviluppatori di accrescerne la ricchezza della piattaforma. Uno dei cardini di questo successo è la facilità di accesso al codice sorgente, utile per velocizzare la comprensione delle funzionalità e svilupparne le estensioni. In qualità di principale promotore di Java, Sun incarna una "filosofia aperta" dello sviluppo, attraverso il Java standards process (si veda JCP.org) e l'implementazione – infatti è possibile già adesso reperire i sorgenti di 1700 progetti su Java.net e NetBeans.org. Credo che Sun sia unica nella capacità e volontà con cui incoraggia i suoi sviluppatori a utilizzare sorgenti aperte.

ioP: Quali sono le tre migliori argomentazioni che utilizzerebbe per convincere uno sviluppatore Java a scegliere Java Studio?

C.A.:

1. Se ha bisogno di apprendere velocemente un codice, può utilizzare in tempo reale l'engineering bidirezionale UML
2. Se vuole rivedere qualche parte del codice con un collega, può usare le caratteristiche di collaborazione istantanea
3. Se vuole sviluppare del codice rapidamente, può utilizzare le caratteristiche di generazione e profiling interne al prodotto.

   E può iniziare immediatamente, rilassandosi mentre guarda il video sulle caratteristiche generali, per poi scaricare il tool gratuitamente ed effettuare il test

ioP: ...e cosa direbbe per convincere uno sviluppatore C# a tornare a Java?

C.A.: Ho già detto qualcosa sulla potenza degli strumenti e sulla community che fa capo a Java. Tuttavia vedo che molti sviluppatori sono accomunati dalla voglia di creare modelli di applicazioni ad alte prestazioni in grado di integrarsi solidamente in ambienti aziendali. In pratica tutti vogliono disporre di strumenti facili da usare e facili da apprendere e che in seguito questi stessi strumenti siano di aiuto per migliorarne la conoscenza e le capacità.
Poiché diventare produttivi usando un nuovo tool richiede un grande investimento in termini di tempo, i programmatori vogliono uno strumento che sia disponibile su più sistemi operativi e che abbia una community forte alle spalle, che supporti una piattaforma dinamica e che disponga di una documentazione adeguata con un numero sufficiente di esempi.

> **"Ancora una volta i vantaggi della piattaforma Java sono da ricercare nella creazione di servizi fruibili dai clienti indipendentemente dalla piattaforma che utilizzano"**

In poche parole gli sviluppatori vogliono un tool che li assista nello sviluppo del codice e che li aiuti a risolvere i problemi rapidamente, senza doversi preoccupare di dove questo codice dovrà girare e con una marcata propensione per la condivisione delle informazioni.

ioP: Potrebbe indicarci le maggiori differ Io credo che il vero punto di forza consista nell'aver fornito un ricco insieme di funzionalità già testate per lavorare insieme – mentre stiamo progettando vogliamo doverci confrontare con i soli problemi legati al progetto e non con quelli derivati dall'ambiente. Con Java Studio Enterprise, i vostri strumenti e il vostro progetto in esecuzione sono tutti testati insieme; abbiamo incluso anche un supporto nel caso abbiate bisogno di assistenza professionale. Inoltre, poiché sia Java Studio che NetBeans si basano sulle componenti standard Swing, vi renderete conto che i vostri strumenti funzioneranno perfettamente con il vostro sistema operativo preferito.

ioP: Cosa vede nel futuro dello sviluppo di Java?

C.A.: La principale caratteristica di Java, dovuta alla community che ne è la base, è che la piattaforma si evolve nelle direzioni che gli sviluppatori ritengono più opportune. Per uno sviluppatore è di grande conforto sapere che la piattaforma è costantemente alla ricerca di soluzioni per problemi che vanno dalla computazione in tempo reale all'integrazione con gli ambienti business: abbiamo a disposizione un punto di partenza sicuro per ogni nostra necessità. Tuttavia, vedo che i team di sviluppo distribuito stanno aumentando sempre di più, nella misura in cui gli applicativi aumentano le loro capacità. In questo scenario, i progettisti modelleranno i processi di client business, per poi scambiare questi modelli con il loro corrispondenti esperti di java implementando così il servizio. Il ripetersi di engineering bidirezionale tra questi team specialistici migliorerà la qualità dei componenti dei servizi di applicazione creati e rilasciati nel network.

SOFTWARE SUL CD

APACHE 1.3.33/2.0.54

Il Web Server più usato al mondo

Apache è certamente il web server che conta il maggior numero di installazioni al mondo. Se un tempo la sua fama era dovuta ad affidabilità ed ottime performance, allo stato attuale pur mantenendo queste caratteristiche può vantare di essere parte di un progetto più grande portato avanti dalla Apache Software foundation e che vede l'integrazione forte di più strumenti sotto un'unica logica. Così ad apache web server si affiancano moltissimo moduli che ne estendono e ne completano le funzionalità andando a contrastare quell'omogeneità che da sempre è la forza degli strumenti analoghi portati avanti da Microsoft. Se avete bisogno di un Web Server perfettamente integrato con PHP e Mysql, Apache è quello che fa per voi.

Directory: /Apache

CASTOR 0.9.6

Un tool di persistenza dei dati leggero

Di Castor parliamo abbondantemente nel bell'articolo di Massimiliano Bigatti pubblicato in questo stesso numero. Si tratta di un tool che consente di mappare righe, colonne e campi di database SQL in corrispondenti classi ed oggetti java. Una valida alternativa ad Hibernate quando si vuole usare un tool senza troppe complicazioni e senza la complessità di Hibernate. Certamente utile per accellerare lo sviluppo di applicazioni che fanno largo uso di DB, altrettanto certamente meno complesso e per certi versi meno potente del fratello maggiore: Hybernate.

Directory: /Castor

COMMONS COLLECTION 3.1

Aumenta la velocità di sviluppo delle applicazioni Java

Le Common Collections sono un set di API che fornisce largo supporto ai problemi della programmazione giornaliera, il loro scopo infatti è fornire un set di classi che consentono di astrarre la programmazione ad un livello superiore mettendo a disposizione del programmatore alcuni strumenti che sono stati in precedenza ricavati dal linguaggio base. Ad esempio le Common Collection mettono a disposizione le interfacce Set e SortedSet che espongono metodi utili per lavorare con le collezioni.

Directory: /CommonCollections

COMMONS HTTPCLIENT 3.0

Accedere al web via Java

Il protocollo HTTP è probabilmente il protocollo più significativo usato su Internet attualmente. Se una volta si poteva considerare come oggetto della programmazione il solo sviluppo di applicazioni Standalone, attualmente la tendenza è quella di sviluppare programmi in grado di fornire un'interfaccia significativa verso il web.

Non solo ma il protocollo http è utilizzato come mezzo di trasporto per una gran mole di dati, vedi ad esempio i Web Services. Il package Java .NET fornisce funzionalità di base per accedere a questo genere di applicazioni ma non offre una sufficiente flessibilità.

Le API *Jackarta Commons HttpClient* offrono un superset di clasi e metodi che aggirano le limitazioni di Java .NET operando in maniera tale da rendere tutte le caratteristiche del protocollo HTTP completamente fruibili dal linguaggio di Sun.

Directory: /CommonHttpClient

COMMONS NET 1.4.0

Internet lato client sotto controllo

Le librerie Jakarta Commons Net implementano il lato client di moltissimi protocollo di base. Lo scopo delle librerie è fornire un livello di astrazione unificato per l'accesso a qualunque tipo di protocollo. L'idea è che il programmatore debba conoscere un solo insieme di classi e metodi attraverso i quali può gestire qualunque tipo di connessione lato client senza per questo doverne conoscere i dettagli di fondo.

Directory: /CommonNet

COMMONS ORO 2.0.8

Espressioni regolari anche in Java

Le regular expression sono una risorsa importante, che può risolvere in pochissimo tempo una quantità enorme di problemi di programmazione legati alla gestione del testo. Il linguaggio principe per l'uso delle Reegular Expression è Perl ma proprio grazie all'uso delle Common-Oro, la stessa potenza si riesce ad ottenere con Java. Si tratta infatti di librerie che consentono di manipolare il testo ad alto livello fornendo funzioni che garantiscono una notevole flessibilità. In particolar modo il supporto alle Regular Expression è piuttosto potente.

Directory: /CommonsOro

COMMONS VFS

Supporto universale a qualunque tipo di File System

Ne parliamo in questo stesso numero

DEVPHP

Un editor PHP che ci semplifica la vita grazie alle sofisticate funzioni che espone

Tutti sanno che è possibile creare una pagina PHP utilizzando semplicemente il notepad. È altrettanto vero che utilizzare uno strumento evoluto come DEVPhp aiuta moltissimo nella stesura del codice. Ovviamente la parte da leone la fa la syntax hihlighting ma anche il debugger, come le altre facilities messe a disposizione dei programmatori aiutano non poco a sviluppare pagine sintatticamente corrette.
Inoltre c'è da dire che DEVPhp è gratuito, particolare non trascurabile se si pensa che per questo linguaggio di programmazione gli editor gratuiti di buon livello qualitativo si contano sulle dita di una sola mano.

Directory: /DevPHP

› CREIAMO UNO SCRIPT

1 Iniziamo creando un nuovo script dal menu "file/new/php script". Verrà creato un file "Untitled1.php" che successivamente potremo salvare in un punto qualunque dell'hard disk e rinominarlo.

› SINTASSI E FORMA

2 Nella creazione dello script si noti la sintax highliting del codice e la code complexion attivabile premendo contemporaneamente i tasti ctrl e barra spaziatrice, queste due funzioni aiutano enormemente il programmatore.

› CONTROLLO DEI RISULTATI

3 Abbiamo scelto di visualizzare la preview del risultato direttamente nel browser digitando l'indirizzo nella barra e utilizzando un nostro web server. In alternativa è possibile usare una preview interna.

nel bell'articolo di Federico Paparoni. Si tratta di librerie che astraggono il programmatore dal dover conoscere i dettagli di implementazione di un File System, in particolare consentono di trattare file system remoti come se fossero locali alla nostra macchina.
Nell'articolo di Federico Paparoni si spiega come sfruttare queste caratteristiche delle librerie Common VFS per utilizzare internet come un grande Hard Disk virtuale da sincronizzare con le risorse locali al nostro computer.

Directory: /CommonsVFS

DBAMGR 2K

Il gestore di database MSDE

Completamente italiano questo progetto e precisamente di Andrea Montanari, ma molto apprezzato anche all'estero si tratta di una console amministrativa alternativa per Microsoft MSDE 1.0 e MSDE 2000 scritta in Microsoft Visual Basic 6.0.
Si tratta di uno dei pochi strumenti disponibili per la gestione visuale dei fratelli minori di SQL Server che per loro natura non sono dotati di una console amministrativa.

Offre funzionalità del tipo: gestione della sicurezza a livello di database, tabelle e vista.
La gestione delle tabelle supporta l'inserimento di nuovi campi e molto altro ancora.
Certo non aspettatevi di potere usare i diagrammi, ma sicuramente si tratta di uno strumento molto interessante se intendete avvalervi di MSDE piuttosto che di MS SQL Server.

Directory: /DBAmgr2K

DEV C++ 4.9.9.2

Il più amato dai programmatori C++

Uno degli IDE più utilizzato da chi programma in C++. In questo numero lo abbiamo usato più di una volta, sia per realizzare alcuni degli express sia nell'articolo relativo alle WX Widgets se pure in una forma estesa grazie all'uso del package per la programmazione delle WX.

È dotato di Syntax Highlighting, code completion, in alcuni casi è possibile

ECPLIPSE SDK 3.0.2

L'IDE di programmazione universale

Eclipse è diventato in breve tempo uno dei prodotti più usati dai programmatori di tutto il mondo. Si tratta ovviamente di un ambiente di programmazione. La prima caratteristica da mettere in evidenza è che l'intero eclipse è scritto in Java e questo lo rende disponibile con funzionalità omogenee sia su Linux che su Windows. La seconda caratteristica è relativa alla sua architettura a plugin. Se da un lato l'ambiente si presenta di default come destinato alla programmazione Java, con semplici plugin diventa un IDE per PHP, per C++ o per praticamente qualunque altro linguaggio disponibile. Inoltre sempre grazie alla sua architettura a plugin, è possibile aggiungere funzionalità aggiuntive che lo rendono ad esempio un ambiente RAD e Visual nel caso si installi il plugin Visual Eclipse. Un must assolutamente da provare.

Directory: /Eclipse

› UN NUOVO PROGETTO

1 Iniziamo dal menu file, selezionando File/New/Project apparirà il wizard per la creazione di una nuova applicazione Java. Selezioniamo "Java Project" e andiamo avanti con Next.

› DIAMOGLI UN NOME

2 Diamo un nome al progetto e scegliamo inoltre se desideriamo che venga creato all'interno dello spazio di default usato da Eclipse, oppure in una directory di nostra scelta.

› LE LIBRERIE

3 Dalla Tabsheet "Libraries" scegliamo le eventuali librerie da aggiungere all'applicazione. È possibile aggiungere dei Jar esterni come delle librerie incluse nel J2SE

addirittura utilizzarlo come strumento di programmazione RAD e Visuale.

Directory /DevCPP

ENCACHE 1.1

Una cache per le applicazioni Java

Di Encache parliamo approfonditamente in questo stesso numero nell'articolo del nostro Daniele de Michelis. Encache è una libreria java che consente di inserire funzionalità di caching all'interno di un'applicazione. Il meccanismo come potrete intuire è piuttosto interessante, perché consente di velocizzare in modo notevole i vostri programmi. Immaginate per esempio tutte le applicazioni che accedono in una qualche misura a dati su ub DB. Un meccanismo di cache consente di evitare l'accesso continuo al database magari remoto per spostarlo invece su una cache locale, ovviamente questo comporta dei problemi nella difficoltà di implementazione del meccanismo di cache e nelle scelte progettuali da compiere. Ogni problema viene risolto da encache che consente di occuparsi della parte più strettamente programmativa senza doversi dedicare alla gestione della cache.

Directory: /encache

IRRLICHT 0.10

La libreria per lo sviluppo rapido di VideoGames

Ad irrlicht dedichiamo costantemente spazio sulla nostra testata.

Si tratta di una libreria veramente molto efficace per lo sviluppo di videogiochi.
Consente di accedere facilmente a funzionalità per la gestione del 3D, per l'implementazione dei parametri d'animazione, per l'applicazione delle texture, per la generazione di effetti complessi.
Si tratta insomma di una libreria decisamente utile per chi vuole iniziare a programmare da zero i propri videogiochi come per chi invece ha bisogno di funzionalità particolarmente avanzate.
Quella che vi presentiamo è la versione 0.10, rilasciata da appena un mese e già ai vertici per quanto riguarda la diffusione

Directory /irrlicht

LAZARUS 0.9.6

Quasi come Delphi ma Gratis

L'OpenSource è probabilmente la più grande rivoluzione dei nostri tempi. Non c'è programma che non abbia un suo equivalente di tipo OpenSource, così Lazarus è il clone OpenSource di Delphi.
Gira tranquillamente sia sotto Windows che sotto Linux ed assomiglia in tutto e per tutto alle versioni di

Delphi fino alla serie 3. Certo non ha la complessità del Delphi 2005 enterprise edition ma non ne ha neanche la pesantezza.

Consente di sviluppare applicazioni standalone complete in piena filosofia Visuale e Rad com'è tipico di Delphi che è stato il primo ambiente a proporsi come IDE completamente Visuale e RAD e ancora oggi offre molto più di altri ambienti di programmazione in quanto a sviluppo rapido di applicazioni

Directory: /Lazarus

MYSQL 4.11/5.0.4

Il principe dei database

È innegabile, la coppia MySQL+PHP è la spina dorsale della maggior parte delle Web Application oggi disponibile sul web. MySQL in questa coppia svolge il ruolo di database e lo fa nella maniera migliore possibile.

La sua caratteristica principale è quella di essere molto veloce, le sue altre caratteristiche sono quelle di supportare le transazioni, la ricerca full text, la sicurezza fino al livello di singola cella. Offre inoltre costrutti SQL estremamente funzionali che in qualche caso superano di gran lunga quelli offerti dal concorrente Microsoft Sql Server. Date un'occhiata all'articolo di Gianluca Negrelli ad esempio sulla paginazione su milioni di record e scoprirete come la presenza del costrutto "LIMIT" in MySQL risolve brillantemente l'intero problema che invece è così pressante in MS Sql Server.

Directory: /MySQL

J2SE 1.5.0_03

L'sdk essenziale per programmare in Java

Se siete dei programmatori Java, sicuramente non potete farne a meno. L'SDK contiene la base per la programmazione nel linguaggio di Sun. Questo Upgrade 0.3 contiene alcune importanti modifiche al compilatore che risolvono diversi Bug della versione precedente. In particolare evitano il crash dell'applicazione sotto particolari condizioni, altre importanti modifiche sono state apportate ad AWT e a Swing.

Directory: /J2SE

JAKARTA SLIDE WEBDAVCLIENT

Le librerie Java per il supporto a WebDav

Il concetto dietro cui si basano queste librerie è particolarmente astratto, ma molto utile. In sostanza si tratta di un framework che consente un organizzazione gerarchica dei file che compongono un'applicazione, intesi come file che possono essere distribuiti su File System eterogenei. Oltre a questo tipo di caratteristiche la libreria integra anche funzioni per la sicurezza il versioning e altri servizi utili

Directory: /jakartaslide

JCIFS

Samba e CIFS in un un'unica libreria

Samba è ormai un protocollo diffuso e affidabile. Se prima era solo Linux ad utilizzarlo per la condivisione delle risorse locali, adesso anche Mac OSX utilizza samba come principale protocollo per accedere a cartelle in rete locale e questo ne ha aumentato esponenzialmente la diffusione.

Questa libreria implementa il protocollo Samba al proprio interno e consente di scrivere applicazioni java che utilizzino questo ormai diffuso metodo per l'accesso a file condivisi in rete

Directory: /Jcifs

WX DEV C++

Usiamo una particolare versione di Dev C++ per programmare interfacce in modo visuale e Rad

Directory: /wx-devcpp

> UN NUOVO PROGETTO

1 Clicchiamo su file/nuovo/progetto per iniziare una nuova applicazione. Il winzard successivo ci consentirà di scegliere un modello da cui iniziare.

> SCEGLIAMO IL MODELLO

2 Fra i tanti disponibili scegliamo di creare un'applicazione windows usando C#. Lo scheletro che ci verrà proposto corrisponderà alla base di una form vuota.

> DISEGNIAMO

3 Dalla tabsheet in basso scegliamo "Disegna". Ci apparirà la form su cui trascinare i controlli che riempiono la barra sinistra del progetto.

PHP 4.3.11 - 5.0.4

Il linguaggio di internet

Se seguite ioProgrammo o più semplicemente siete dei programmatori Web, o ancora molto più semplicemente navigate su Internet, non potete non sapere che cosa è PHP. Si tratta del linguaggio con il quale sono sviluppate la maggior parte delle applicazioni internet esistenti. Quasi tutto il software per il web si regge su PHP.
La curva di apprendimento è bassissima, le funzionalità esposte elevatissime, certamente se avete intenzione di sviluppare per il web non potrete fare a meno di provare anche questo linguaggio come base per le vostre applicazione

Directory: /PHP

ULTIMATE ++

L'IDE più innovativo per C++

Se DEV C++ rappresenta ormai uno standard per i programmatori C++, Ultimate ++ si sta dimostrando una ottima alternativa. Leggero, veloce, dotato di alcune estensioni che lo rendono in parte RAD, viene distribuito con il compilatore MinGW.

È dotato naturalmente di tutte le caratteristiche di un buon IDE, tuttavia dispone di un'organizzazione che lascia rapidamente intendere che da questo ambiente dovremo attenderci delle grosse novità nel futuro, sia in termini di prestazioni che di completezza.

Directory: /Ultimatepp

SHARPDEVELOP 1.0.3

L'ambiente alternativo per la programmazione C#

Per chi non vuole affrontare i costi di Visual Studio.NET, per chi comunque vuole disporre di un ambiente abbastanza leggero per potere programmare le proprie applicazioni utilizzando C#, SharpDevelop sembra essere l'unica alternativa.

Da poco è possibile utilizzarlo anche per programmare in VB.NET, tuttavia la sua caratterizzazione è rivolta prevalentemente a C#. Si tratta di un prodotto ormai consolidato, rapido, visuale, economico e semplice da usare

Directory: /SharpDevelop

PYTHON 2.4.1.

Il linguaggio emergente

Un linguaggio di scripting, multipiattaforma, orientato agli oggetti. Tre caratteristiche che rendono questo linguaggio piuttosto appetibile. Non per niente tutte le classifiche mondiali sulla diffusione dei linguaggi di programmazione lo individuano come quello maggiormente in ascesa. Python è elegante, estensibile, con una curva di apprendimento non elevatissima.
Viene utilizzato per programmare moltissimi tool di gestione dei sistemi Unix, ma sta trovando una rapida applicazione anche sui sistemi windows e nello sviluppo di applicazioni crossplatform.

Directory: /Python

THING 0.1

L'editor di Thinlet

Thinlet è una libreria java che consente di creare interfacce grafiche utilizzando XML. Le interfacce così create vengono parserizzate dalla libreria e producono in output l'applicazione corretta.
Thing è un editor XML per lo sviluppo rapido è visuale di interfacce compatibili con Thinlet. Si tratta di un editor scritto in java, perciò perfettamente funzionante sia con Linux che con Windows.

Ha dalla sua una certa pesantezza nell'esecuzione ma una volta presa confidenza con la logica di funzionamento l'accoppiata Thing +Thinlet si mostra decisamente potente e garantisce un abbattimento notevole dei tempi di sviluppo.

Directory: /Thing

TOMCAT 5.5.9

L'application server per JSP

Se siete dei programmatori JSP avete senza dubbio bisogno di Tomcat per testare e utilizzare le vostre applicazioni. È senza dubbio un server web ovvero può funzionare da server web ma è soprattutto un Application Server che vi consente di utilizzare la tecnica delle servlet o delle JSP per creare applicazioni Web con una logica business e con il linguaggio Java a fare da asse portante.

Directory: /Tomcat

PIRCBOT

L'irc programmabile in Java

Ne parliamo nell'articolo di Federico Paparoni. Si tratta di un framework per la programmazione di irc bot. Chi ha dimestichezza con la rete irc sa già cosa sono gli eggdrop e come funzionano. Piccoli software in grado di reagire ad eventi che si svolgono all'interno di una chat e di "governare" un canale sulla base della volontà del suo moderatore.
Mentre Eggdrop è un bot già programmato, sviluppato in ogni sua parte anche se estendibile in TCL, Pircbot viceversa è un'insieme di classi Java che consentono di realizzare facilmente e in autonomia un proprio bot dotato di comandi, comportamenti e funzioni completamente aderenti alla volontà di chi lo realizza.
Per chi ama la rete IRC si tratta vera-

mente di una “perla” da provare assolutamente.
Directory: /Pircbot

JAKARTA COMMONS LOGGING

Le api essenziali per il Debugging

I log svolgono un'attività essenziale nella manutenzione di un'applicazione sia in fase di test che in fase di rilascio. Solo attraverso ai log si riesce ad ottenere un quadro preciso di quali eventi hanno portato a un problema nell'utilizzo di un determinato software. La Jakarta Commons Loggin offre funzionalità di alto livello per la gestione dei log di un'applicazione, consente di creare nuovi formati di Log come di riutilizzare quelli gia esposti in altri componenti come HttpClient o DBCP.
Directory: /logging

SNARLI

La libreria Java per lo sviluppo di reti neurali

Sogno da sempre di ogni appassionato di programmazione e di tecnologia in generale è quello della creazione di una generazione di robot pensanti. Sogno ambizioso e non privo di preoccupazioni e rischi, ma pur sempre affascinante. In questo numero di ioProgrammo, ci conduce nell'affascinante mondo della costruzione di reti neurali.
Una rete neurale è una simulazione dell'intricata rete di relazioni che compone il cervello umano e che tramite l'applicazione si stimoli sensoriali produce il comportamento intelligente. Strumento pratico per implementare quanto descritto da Andrea Galeazzi è Snarli: una libreria Java sviluppata da Simon D. Levy del dipartimento di informatica dell'università di Washinghton e Lee. Snarli mette a disposizione una serie di metodi che aiutano il programmatore a creare e addestrare reti neurali simulate.
Directory: /Snarli

XERCES 1.4.4

Il parser XML per Java e C+

Xerces è una libreria Java che fornisce una serie di classi per il parsing e la generazione di file XML. Il parser è disponibile sia per Java che per c++, implementa gli standard del W3C Xml e i livelli uno e 2 del DOM, inoltre supporta SAX nella sua versione 2.Xerces può essere utilizzato con dei wrapper per funzionare ad esempio anche in perl e infine esiste un wrapper che rende Xerces compatibile con il parser MSXML
Directory: /Xerces

XALAN

Il processore XSLT per Java e C++

Alcuni di voi avranno sentito parlare di XSLT che consente di trasformare un documento in XML in un interfaccia Html, oppure di XPath il linguaggio che consente di ricercare dati all'interno di un documento XML. L'unione delle due tecniche conduce spesso ad effetti sorprendenti. Xalan è un'implementazione java di un parser XSLT. Implementa gli standard W3C XSLT e usa il Bean Scripting Framework per fornire estensioni Java ed Sql.
Directory: /xalan

OPENRPT 1.1.1 BETA WIN32

Report per tutti i gusti

OpenReport è un sistema di reportistica completa ed OpenSource dotato di interessanti caratteristiche.

Desideriamo presentarvelo nonostante si tratti ancora di una beta proprio perché le sue potenzialità sono davvero interessanti. Si tratta di un sistema WYSIWYG che salva la definizione del report in formato XML Già da sole queste due caratteristiche basterebbero da sole a rendere il sistema particolarmente interessante, a tutto questo c'è da aggiungere che OpenRpt gestisce molti tipi di codice a Barre ed è multipiattaforma, funziona tranquillamente su Windows, Linux, Mac OS X, Xbsd, Solaris, Aix, HPUX.

Unico neo per questo sistema è che il supporto nativo ai Db è presente solo per postregreSQL tuttavia con poche modifiche si può rendere compatibile con DB2, Oracle, SQL Server, MySQL.
Directory: /openrpt

DEV C++ PACK

Tre package per estendere Dev C++

Dev C++, il noto ambiente che presentiamo spessissimo su ioProgrammo è estensibile tramite plugin o package, ve ne presentiamo tre, due dei quali fanno diventare Dev C++ RAD e visuale tale che possa essere usati con la libreria WxWidgets che presentiamo in questo stesso numero. Il terzo ancora da utilizzare con le wxWidgets fornisce alcune caratteristiche per la gestione delle immagini. Nel complesso tre strumenti fondamentali da usare per programmare in C++ usando le librerie universali wxWidgets.
Directory: /devpack

WXDEVCPP

un Dev C++ potenziato

Un'installazione semplificata che vi consenta di usare da subito Dev C++ con le estensioni dedicate alle wxWidgets tali che lo rendono un ambiente rad e pronto per essere usato per la produzione di applicazioni multipiattaforma. Ovviamente questa installazione funziona in ambiente Windows. Ma Dev C++ così configurato rappresenta una straordinaria alternativa a sistemi da costi commerciali molto più elevati e non per questo dotati di caratteristiche molto più evolute.
Directory: /wxWidgets

Pocket Builder

Arriva da Sybase una soluzione per lo sviluppo RAD di applicazioni dedicate al mondo Mobile. La programmazione per Pocket PC diventa rapida e immediata, le applicazioni Wireless a portata di mano

Pocket Builder è l'ambiente di sviluppo RAD proposto da Sybase per la programmazione di sistemi mobili Basati su Microsoft Pocket PC 2002 e Windows Mobile 2003. Le caratteristiche sono piuttosto interessanti. Intanto si tratta di un ambiente di programmazione RAD, perciò l'intento è quello di consentire a chiunque non abbia un livello alto di esperienza sulla programmazione di dispositivi mobili, comunque, di creare una propria applicazione in tempi ragionevolmente brevi. L'idea è quella di usare una programmazione basata sul Drag & Drop degli oggetti piuttosto che sulla reale implementazione del codice, tecnica che nel tempo si è dimostrata particolarmente valida. Il secondo punto a favore di Pocket Builder deriva dall'innata caratteristica di Sybase di creare software orientato alla gestione dei dati. Pocket Builder non sfugge a questa regola, e dispone di estensioni tali che lo rendono particolarmente indicato per sviluppare applicazioni per dispositivi mobili che facciano uso di database. In particolare è disponibile nella stessa suite il prodotto SQL AnyWhere Studio che è sicuramente un supporto di grande validità allo sviluppo di software DB Oriented. Esistono diversi Wizard che consentono di sviluppare velocemente applicazioni di database, in particolare il MobiLink consente con pochi passi di ottenere applicazioni completamente funzionanti.

APPLICAZIONI PRONTE PER LA VOCE

Da un tool nato per creare software destinato al mobile non ci si poteva non aspettare tutta una serie di estensioni dedicate all'uso della voce o comunque a modi di comunicare nuovi quali ad esempio gli SMS .Questo tipo di servizi sono facilmente implementabili tramite Pocket Builder rendendolo particolarmente adatto alla creazione di prodotti che integrano in un'unica soluzione tutte le estensioni multimediali possibili. A questa logica non sfuggono l'integrazione con le camere digitali che da qualche tempo sono accessorio indispensabile per ogni smartphone che si rispetti. Infine sono previste estensioni per la gestione del GPS e dello Scanner.

ALTRE CARATTERISTICHE

Pocket builder supporta direttamente la stampa, e integra la possibilità di reportistica accurata partendo dalla vostra applicazione. Si integra facilmente con le applicazioni già esistenti come il Calendario, o la gestione degli appuntamenti e dei task. Infine è pronto per la connettitività wireless. La sicurezza viene gestita tramite la possibilità di firmare il propiro codice con un certificato digitale.

CONCLUSIONI

Pocket Builder rientra in quella fascia di IDE RAD dedicati alla programmazione di nuova generazione che vede non più le applicazioni standalone come target dello sviluppo, supera abbondantemente la programmazione web e si dedica direttamente a quel settore che da molti analisti è definito come la nuova frontiera dello sviluppo, ovvero il mobile. I suoi punti di forza sono prima di tutto, relativi alla sua natura di ambiente RAD e in secondo luogo all'alta integrazione con la gestione dei dati derivante direttamente da un marchio che in questo campo offre delle garanzie certe: Sybase. Se siete interessati a programmare applicazioni per il mobile dovete almeno provarlo.

Fox Micro Linux System

In 66x72mm un condensato ad alta tecnologia. Connessione ethernet, sistema operativo linux, due porte USB, seriali, Scsi e 19 ingressi/uscite general purpose. Gli manca solo la parola...forse

I sistemi embedded si stanno sempre più diffondendo. Si tratta di piccole macchine per lo più dedicate a compiti specifici utili ad esempio in sistemi di automazione industriale. Fino a qualche tempo fa il leader in questo settore era il mitico microcontrollore Pic16f84, da qualche tempo alla famiglia dei Pic si è affiancata una nutrita schiera di device, dei veri e propri PC dotati di sistema operativo e di tutte le caratteristiche di un computer vero e proprio ma con dimensioni tali da renderli appetibili come sistema embedded.

LA DOTAZIONE

Il cuore della scheda è un processore Axis ETRAX 100LX 32 bit, RISC, 100 MHz, la dotazione di memoria è di 16 Mb ram con 4mb flash. L'accessoristica è di tutto rispetto, due porte USB 1.1, una scheda ethernet 10/100, il resto è lasciato alla vostra fantasia. I due slot da 40 piedini contengono il necessario per implementare 3 seriali, 2 Bus IDE, 2 Bus Scsi, un'interfaccia parallela e un nutrito numero di entrate uscite general purpose. Questa dotazione rende la Fox Microlinux Systems particolarmente appetitosa come soluzione dedicata alla costruzione di sistemi embedded. Dal punto di vista software, il sistema operativo è pilotato dalla versione 2.6 del kernel di Linux. Nella busybox sono presente tutte le componenti essenziali per funzionare immediatamente. È già configurato un server Web un server Ftp, il servizio telnet, ssh.

COME FUNZIONA?

È sufficiente alimentare il sistema e attaccarlo alla propria Lan per vederlo funzionare. Ovviamente è necessario connettersi con uno dei sistemi sopraindicati per poter effettuare configurazioni personalizzate come ad esempio il cambio dell'indirizzo IP. Inizialmente il sistema è configurato sull'indirizzo 192.168.0.90. Accedendo via interfaccia web è possibile variare i parametri direttamente dal browser. La programmazione avviene per mezzo di un SDK disponibile sul sito *http://www .acmesystems.it*. È possibile programmare direttamente in C e poi effettuare il deploy dell'applicazione verso la scheda.
Una prima occhiata agli esempi di codice allegati all'SDK mostra come programmare questo genere di apparecchi sia incredibilmente più semplice che imparare l'assembler del PIC.

Fig. 1: *Uno schema generico delle possibili connessioni offerte dalla piedinatura del FOX*

CONCLUSIONI

Si tratta di una soluzione estremamente efficace in tutte quelle condizioni dove programmare un PIC risulta scomodo e difficoltoso. Il costo dell'apparecchio si aggira intorno ai 138 euro, ma non ci sono costi software da sostenere, visto che l'sdk è gratuito e la programmazione può avvenire direttamente da un sistema Linux.
Infine poter programmare partendo da un linguaggio ad alto livello come il C sicuramente offre dei vantaggi non indifferenti in termini di costo dello sviluppo.

Processi concorrenti con i semafori

Che cosa succede quando due applicazioni tentano di accedere alla medesima risorsa? I semafori introdotti da Dijkstra forniscono una soluzione e rappresentano un ottimo punto di partenza per il problema

Al metodico cammino che ci sta portando alla comprensione della programmazione concorrente si aggiunge un nuovo passo: i semafori. Ricordo che perché due o più processi cooperino, ossia possano essere eseguiti in modo concorrente è necessario che venga correttamente gestita una porzione di codice condiviso. In tale regione chiamata sezione critica ogni processo condivide delle variabili. Quando un processo elabora ed utilizza tale regione non bisogna permettere l'accesso agli altri processi. Si parla di mutua esclusione. I metodi di gestione della mutua esclusione visti nello scorso appuntamento risultano macchinosi e poco funzionali nel caso di problemi complessi. Una svolta alla risoluzione di tale problema è stata apportata da Dijkstra che ha introdotto nuovi strumenti di sincronizzazione tra processi: i semafori.

SEMAFORI

Un semaforo è una variabile *S* che eccetto la fase di inizializzazione è manipolata da due sole operazioni atomiche: *P* e *V.* Esaminiamo le definizioni delle due funzioni.

```
P(S):   while (S<=0) do;
        S:=S+1;
V(S):   S:=S+1
```

Cosa si intende per atomiche? Che le operazioni svolte nelle due funzioni presentate non possono essere interrotte e devono essere eseguite in modo indivisibile. Nel caso di *P* non si possono interrompere le due istruzioni con altre. Se un processo modifica il valore del semaforo, in quel intervallo di tempo nessun altro processo può modificare lo stesso semaforo. Osserviamo adesso come questo nuovo costrutto sia usato nella gestione delle sezioni critiche. Poi esamineremo nei particolari l'implementazione. Si suppone che *n* processi condividano un semaforo di nome *mutex,* che ha valore iniziale pari a 1. Ogni processo gestisce la sezione critica attuando il ciclo infinito proposto di seguito:

```
repeat
        P(mutex)
        <regione critica>
        V(mutex)
        <codice rimanente>
until false
```

Fig. 1: ***Grafo di precedenze per l'esecuzione di sette istruzioni***

Si tratta di una sequenza conosciuta. Questa volta la sezione critica è protetta dalla presenza di un semaforo che impedisce al processo di entrare se mutex è occupato. Una volta terminato lo sfruttamento della regione la si libera con *V*. Esaminiamo altri esempi per i quali i semafori sono utilizzati con profitto. Consideriamo il caso in cui si abbiano due processi *P1* e *P2* eseguono rispettivamente due istruzioni *S1* e *S2*. Supponiamo di avere come vincolo che l'esecuzione di *S2* debba avvenire soltanto dopo che *S1* sia completamente terminata. Il processo può essere sincronizzato mediante l'uso di un semaforo. Sia *sincr* il semaforo citato con valore iniziale 0, la soluzione si otterrà facilmente modificando di poco i due processi *P1* e *P2*.

```
(* P1 *)
```

```
S1;
V(sincr);
(* P2 *)
P(sincr);
S2;
```

Poiché *sincr* è inizializzato a 0, *P2* eseguirà il suo statement *S2* soltanto dopo che verrà superata la funzione *P*, ossia quando *P1* invocherà *V(sincr)*, quindi dopo *S1*. Ricordo che la *V* incrementa la variabile *S*, cosicché per *P*, la condizione *S<=0* risulterà falsa.
Si tratta di un caso più complicato ma che possiamo opportunamente risolvere adottando i costrutti *cobegin* e *coend*. Ecco come:

```
var sema,semb, semc, semd,seme, semf,
                                        semg:semaforo;
< Inizializzazione dei semafori>
begin
     cobegin
     begin S1; V(sema); V(semb) end;
     begin P(sema); S2; V(semc); V(semd); end;
 begin P(semb); S3; S4; V(seme) end;
 begin P(semc) S5; V(semf) end;
 begin P(semd) S6; V(semg) end;
 begin P(semf); P(semg); P(seme); S7end;
coend;
end;
```

Ogni qual volta vi sono due istruzioni eseguibili concorrentemente si lanciano due *V*, che in due distinte istruzioni vengono raccolte dalle funzioni *P*. La realizzazione del grafo con questo nuovo metodo dimostra tra l'altro che non è indispensabile il costrutto *fork* e *join* per la risoluzione di tale tipologia di problemi.

IMPLEMENTAZIONE

Alcune questioni sono rimaste in sospeso, come ad esempio la modalità di inizializzare un semaforo. Inoltre, bisogna studiare il modo di implementare i semafori. Dall'analisi della situazione emerge come maggiore svantaggio dell'approccio alla mutua esclusione l'attesa nelle situazioni di occupato. Se un processo è nella sezione critica, un altro processo che tenta di entrare nella stessa sezione è costretto ad un ciclo a vuoto. Un loop che non consente al processo di fare altro. Nella programmazione reale ciò definisce un grande limite. Per superare il problema bisogna modificare l'implementazione delle due funzioni *P* e *V*, rispetto alla definizione base proposta. Trovare dei meccanismi che eliminino o quanto meno minimizzino tale problema conosciuto come: dell'attesa limitata. La fase di attesa deve essere sostituita dalla possibilità per il processo di riprendere. Si tratta di far passare il processo che in questa situazione si trova allo stato di blocca-to a quello di pronto. L'implementazione di tale evento avviene, innanzitutto, modificando il tipo semaforo che sarà adesso associato ad una variabile strutturata non omogenea. Ecco come può essere dichiarato il record.

```
type semaforo=record
            valore:integer;
            l: <lista di processi>;
end;
```

Quindi un semaforo sarà caratterizzato da un valore, che ha il significato che conosciamo, e da una lista di altri processi. Questa ultima non è stata di proposito descritta nei dettagli, ad ogni modo si tratta di una lista a puntatori. Una lista di *PCB (process control block)* descrive al meglio il secondo campo. Insomma una lista in cui ogni nodo presenti descrittori di processi. Quando un processo deve attendere ad un semaforo esso viene aggiunto alla lista di processi e ovviamente si incrementa il valore. La funzione *V* decrementerà il valore e rimuoverà il processo liberato. Alla luce di questa nuova struttura dati l'implementazione delle due funzioni si trasforma come segue:

```
P(S):      S.valore:=S.valore-1;
           if S.valore<0
           then begin
                        <aggiungi il processo a S.L>
                        block
                 end;
V(S):      S.valore:=S.valore+1;
           if S.valore<=0
           then begin
                        < rimuovi il processo da S.L>
                        restart(P)
                 end;
```

L'operazione *block* sospende il processo che lo invoca. L'operazione restart riavvia l'esecuzione del processo bloccato da *P*. Con questa definizione il valore di *S* potrà essere negativo a differenza della definizione classica quando ciò non era possibile.
Il valore negativo indica il numero di processi in attesa al semaforo. Ciò è la conseguenza del decremento introdotto nella funzione *P*. L'aggiornamento della lista avviene seguendo una filosofia *FIFO (First in first out)* tipica delle code, ossia il primo ad entrare e il primo ad uscire. Ad ogni modo non è un elemento rilevante la strategia usata per la produzione e la gestione della lista dei processi. Per cui è possibile trovare in alcune implementazioni la gestione della lista con code a priorità e in rari casi con pile.
L'elemento centrale nella realizzazione dei semafori è il modo in cui si garantisce l'atomicità di esecuzione. Bisogna assicurare che non ci siamo mai due

Utilizza questo spazio per le tue annotazioni

ESECUZIONE CONCORRENTE

L'esecuzione concorrente tra processi è garantita quando:

- **Si ha mutua esclusione: se un processo è nella regione critica allora nessun altro processo ci si può trovare;**
- **Non ci sono processi in esecuzione nella regione critica e altri processi ne fanno esplicita richiesta, allora uno di essi deve entrare. La decisione non può spettare ai processi che si trovano nella parte rimanente. Inoltre, non si può posticipare indefinitamente nel tempo tale decisione.**
- **Vi è attesa limitata: deve esistere un limite superiore di numeri di processi che entrino nella regione critica prima di un generico processo.**

o più processi che eseguano operazioni di P e V dello stesso semaforo allo stesso tempo. Questo è un problema di sezione critica e può essere risolto in due modi. Per processori singoli si può semplicemente inibire gli interrupt durante il tempo delle operazioni di P e V. C'è da dire che alcune operazioni di sistema non sono in grado di sopportare lunghe sospensioni della CPU, come ad esempio la lettura dei dati da un disco; altrimenti i dati si perdono. Si ovvia nel caso specifico al problema con la presenza di buffer. Comunque in definitiva non si tratta di una reale difficoltà considerato che le istruzioni di P sono pochissime, e si realizzano quindi in tempo esiguo. In un sistema a multiprocessore (con più CPU) tale metodo è impraticabile poiché le istruzioni di diversi processi, nella fase di running, possono essere manipolate su diversi processori in modo arbitrario. In tal caso è necessario adottare uno dei conosciuti metodi per la gestione critica con flag (algoritmi 4, 5 e 6 presentati nello scorso numero). Con ciò bisogna sinceramente ammettere che l'effetto indesiderato attesa limitata non è del tutto eliminato. Si è spostata la protezione delle sezioni critiche sulle sole operazioni di P e V, che è meglio delle precedenti soluzioni che si applicavano a tutta la fase di entrata. Le operazioni su P e V sono più corte con conseguente diminuzione di tempi di attesa. Inoltre, è scomparsa l'attesa limitata dall'ingresso alle sezioni critiche. Per approfondire analizziamo un paio di esempi che fanno uso dei semafori.

NOTA

CURIOSITÀ

Le due funzioni *P* e *V* sono anche conosciute come *wait* e *signal*. Anzi la loro prima definizione aveva questo nome. La *P (wait)* indica che bisogna aspettare così come fa un automobile davanti ad un semaforo rosso. Mentre la *V (signal)* indica il lancio di un segnale. Nell'analogia stradale significa che una direzione dell'incrocio è stata servita e il semaforo può diventare verde per altri automobilisti.

PRODUTTORE CONSUMATORE

Il problema che già conosciamo prevede la presenza di due entità. Un produttore che genera dati ed un consumatore che assorbe le produzioni del produttore. Considerando come più flessibile un sistema di scambio asincrono risulta indispensabile la presenza di un buffer. Supponiamo che il buffer sia costituito da n dati. Bisogna prevedere un semaforo che controlli il buffer. A tale semaforo devono riferirsi i due processi i quali non possono per ovvie ragioni accedere in contemporanea al buffer.

Fig. 2: ***Schema del problema Produttore - Consumatore***

Inoltre, il produttore avrà un proprio semaforo che impedirà di continuare le operazioni quando il buffer è pieno, mentre il consumatore analogamente si fermerà quando il buffer è vuoto. Ecco il codice nella sua completezza. Si tralascia la sola dichiarazione del tipo dato che dipende dal caso specifico di utilizzo. Il buffer sarà un array o una lista a puntatori di elementi di tipo dato.

```
..
var pieno, vuoto, mutex: semaforo;
   datop, datoc: dato   ;
begin
   pieno:=0;
   vuoto:=n;
   mutex:=1;
   cobegin
   produttore:
   repeat
        <produce il dato datop>
        P(vuoto);
        P(mutex);
        <aggiunge il dato datop al buffer>
        V(mutex);
        V(pieno);
  until false;
  consumatore:
  repeat
        P(pieno);
        P(mutex);
        <rimuove il dato datoc dal buffer>
        V(mutex);
        V(pieno);
        <consuma il dato datoc>
  until false;
  coend
end.
```

Quando il produttore incontra *P(mutex)* si bloccherà se il semaforo è occupato il che vuol dire che il consumatore sta avendo accesso al buffer.
Ragionamento speculare per il consumatore. Il produttore prima di aggiungere un nuovo dato nel buffer verifica se i due semafori gli danno via libera. Il primo è libero se non occupato dal consumatore. Il secondo è libero purché ci sia qualche posto libero nel buffer. Vuoto è inizializzato a n; ogni volta che il produttore lancia una *P(vuoto)* sottrae un elemento vuoto, quindi ne aggiunge uno occupato al buffer e procede. Si ferma nel caso vuoto sia 0, ossia non ci

QUEUE E STACK

Le due strutture queue (coda) e stack (pila) seguono due diversi metodi di accesso ai dati. Entrambe si possono realizzare con array o liste a puntatori. Nel caso della coda si segue la FIFO (first in first out) che vuol dire che il primo elemento ad entrare nella struttura dati è anche il primo ad uscirne. Per intenderci è il caso di una normale coda di persone ad uno sportello, dove il primo a entrare è il primo ad essere servito. Nel caso della pila si segue il metodo LIFO (last in first out) che indica che l'ultimo ad entrare è il primo a uscire. Il tal caso le due operazioni di inserimento (push) e prelevamento (pop) avvengono dallo stesso lato.

sono posti liberi. Poi ogni qual volta inserisce un nuovo elemento nel buffer manda un segnale al semaforo pieno (una *V*) che vuol dire che sta aggiungendo dati. Se, infatti, quando pieno è ancora 0, il processo consumatore provasse a effettuare una operazione si troverebbe a dover attendere sul semaforo pieno. Basta almeno un solo inserimento del produttore e tale semaforo segna verde. Si riscontra una precisa simmetria tra il codice associato al produttore e quello associato al consumatore. Si può dire che il produttore ha il compito di generare buffer pieni per il consumatore; e che il consumatore ha come compito quello di produrre buffer vuoti per il produttore. Un esempio esplicativo sull'uso dei semafori. Ma analizziamone un altro ancora più intrigante.

I FILOSOFI AFFAMATI

Si tratta di un curioso e importante problema la cui soluzione in ambito concorrente segna un percorso da seguire per una ricca gamma i problemi della stessa tipologia. Anzi per molti programmatori si tratta del paradigma per i problemi sulla sincronizzazione. Descriviamo la scena e gli attori. Si tratta di un banchetto di *n* filosofi che svolgono le due sole attività di pensare e mangiare. Attingono ad un vassoio di riso e poiché sono cinesi usano le bacchette, due per la precisione. Anche le bacchette come i filosofi e i vassoi sono n, per cui un filosofo può mangiare solo se trova libera una bacchetta a destra e una sinistra. È facile l'analogia con il mondo della programmazione. Le bacchette sono risorse e i filosofi processi che consumano risorse.

Fig. 3: ***La scena di riferimento per il problema dei filosofi affamati***

L'idea alla base della soluzione è associare un semaforo ad ogni bacchetta. Si tratterà quindi di definire un array di n semafori. Facendo riferimento al caso in cui il colto banchetto comprenda 5 saggi, si può implementare la soluzione, per il generico filosofo, come segue. Si suppone che tutte le bacchette siano inizializzate a 1.

```
..
var bacchette : array[0..4] of semaforo;
…
(* azione del filosofo i *)
repeat
        P(bacchette[i]);
        P(bacchette[i+1]);
        <mangia>
        V(bacchette[i]);
        V(bacchette[i+1]);
        <pensa>
until false;
```

Il generico filosofo deve attendere che siano libere le due bacchette di sua competenza, di indice *i* e *i+1*. Una volta liberate può mangiare e successivamente riposare le bacchette sul tavolo, azione che corrisponde al richiamo delle due funzioni *V*. Quindi può ritornare nelle sue lunghe e profonde analisi (sic!). A volere essere pignoli i pensatori potrebbero trovarsi nella situazione di non mangiare e non pensare, brutta condizione che dobbiamo evitare ai nostri saggi. Se ogni filosofo prende la bacchetta alla propria destra, ovvero supera *P(bacchetta[i])*, e si blocca sulla successiva, quella a sinistra *P(bacchetta[i+1])* si presenta uno stallo, un *deadlock*. Si ha attesa circolare. Ogni filosofo attende senza poter far nulla. Si può risolvere mettendo in campo alcune idee; esaminiamole:

- Si invita un filosofo in meno;
- Analogamente alla prima idea si aggiunge una bacchetta. Tradotto nella realtà informatica si aggiunge una risorsa (purché non si tratti di uno spreco può essere attuata);
- Un filosofo prima di prendere la bacchetta deve sincerarsi che anche l'altra sia disponibile.
- Soluzione asimmetrica. I filosofi pari prendono prima la bacchetta alla propria destra e poi quella alla propria sinistra. I filosofi dispari fanno il contrario. Potenza dell'ingegnosità.

Le ultime due soluzioni non prevedono modifiche della situazione iniziale.

CONCLUSIONI

Abbiamo visto come i problemi di sincronizzazione sono ben gestiti mediante i semafori. Il percorso anche se comincia a presentare le prime salite ci mostra orizzonti sempre più interessanti e meritevoli di essere osservati e approfonditi. Il sentiero ancora non è alla fine. Vi aspetto per altre piacevoli esplorazioni, tutto nel fantastico mondo della programmazione concorrente e dei sistemi operativi. Alla prossima!!

Fabio Grimaldi

DEADLOCK

Letteralmente punto morto o stallo. Nella programmazione concorrente un insieme di processi è in *deadlock* quando ciascun processo dell'insieme è in attesa di un evento che può essere causato soltanto da un altro processo dell'insieme. Verosimilmente gli eventi sono l'acquisizione e il rilascio delle di risorse.

ON LINE

JAVA ITALIAN PORTAL

Nato dalla passione di un gruppo di ragazzi per l'informatica, JavaPortal si è trasformato nel tempo in un ricco contenitore di articoli tecnici, notizie, link utili riguardo al mondo del linguaggio del chicco di caffè.
http://www.javaportal.it

OPENSKILLS

Esperimento Aperto di Knolewdge condiviso – portale per Sysadmin Linux – Questo è quanto si legge come descrizione del sito OpenSkills. In effetti mai nessuna descrizione fu più azzeccata di questa. OpenSkills è un ricco contenitore di informazioni dedicata agli amministratori di un sistema Unix.
http://openskills.info/index-it.php

DEVELOPERFUSION

Hai bisogno di un pezzo di codice con un esempio rapido che risolva un tuo problema? Ti serve un trucco veloce da applicare? DeveloperFusion è quello che fa per tè. Il sito contiene centinaia di piccoli snippet divisi per linguaggio e argomento.
http://www.developerfusion.com

Biblioteca

BLACK HAT CRIMINI, MISFATTI E TRUFFE SUL WEB

Divertente quanto inquietante questo libro di John Biggs. Divertente perché affronta in modo ironico un problema serio, inquietante perché il problema è talmente serio che un autore delle qualità di John Biggs ha sentito l'esigenza di scrivere un libro sull'argomento. Inquietante anche perché le truffe sul Web sono diventate talmente diffuse che al di là di alcuni casi particolarmente inquietanti non suscitano più nell'utente la perplessità di un tempo, a testimoniare che il Web è diventato ormai un parallelo della realtà e ne ricalca pregi e difetti. Come nella realtà esiste sul web un popolo di criminali e tal volta fantasiosi truffatori che esercita la propria "professione" nei meandri della rete piuttosto che in quelli più tangibili della realtà quotidiana. Black Hat si snoda fra crimini e truffe di tutti i tipi, ripercorrendo puntualmente la storia del crimine informatico e tracciando parallelamente alla sua evoluzione anche quella tecnologica. Il parallelismo fra crimine informatico e avanzamento della tecnologia è d'altra parte tangibile se si pensa alla quantità di patch che vengono diffuse annualmente per difendersi dai moderni truffatori. Una storia di moderni "ladri di biciclette" che in più di un caso non lascia spazio al sorriso.

Difficoltà: Bassa • **Autore:** John Biggs • **Editore:** Mondadori • **ISBN:** 88-04-53739-6 • **Anno di pubblicazione:** 2004 • **Lingua:** Italiana • **Pagine:** 189 • **Prezzo:** € 12,80

TCP/IP PER LAVORARE MEGLIO

Che TCP/IP sia il protocollo di base su cui si fonda l'intero universo del Networking è noto. Tutti i sistemi di comunicazione oggi più diffusi si basano sulla trasmissione dei dati in formato TCP/IP. Ma come funziona TCP/IP? che differenza c'è fra TCP e IP? quali sono e come funzionano i protocolli di livello superiore che si basano su questo standard? A queste domande e a molte altre trovate la risposta nel bel libro di Karanjit S. Silvan e Tim Parker edito da Apogeo. Si tratta di un volume dal notevole spessore in cui si parte dalle basi del protocollo affrontandolo tramite le sue RFC per arrivare nel corso dei capitoli ad affrontare argomenti quali il protocollo SNMP o la sicurezza delle trasmissioni, argomento di grande attualità. Si tratta di un libro veramente interessante per chi non si accontenta di usare in modo meccanico gli strumenti che la tecnologia mette a disposizione ma ne vuole invece comprendere le basi, vuole in un certo senso essere parte della tecnologia e non subirla.Il libro è ben scritto e nonostante la difficoltà nel porre informazioni così complesse scorre con una certa piacevolezza alternando informazioni storiche e di carattere generale alle ben più difficoltose notizie tecniche.

Difficoltà: Alta • **Autore:** Karanjit S. Siyan e Tim Parker • **Editore:** Apogeo • **ISBN:** 88-503-2044-2 • **Anno di pubblicazione:** 2002 • **Lingua:** Italiana • **Pagine:** 862 • **Prezzo:** € 52,00

INTERMEDIATE ROBOT BUILDING

La robotica sta entrando pesantemente nella nostra vita quotidiana e quello che la rende ancora più interessante è che la personalizzazione dei piccoli robot che migliorano la qualità della vita comincia a diventare accessibile anche a programmatori non particolarmente esperti.
Se da un lato è vero che non si può pretendere di affrontare immediatamente la programmazione di un robot per l'automazione industriale, è altrettanto vero che esistono una serie di sistemi programmabili con relativamente poco sforzo e con una certa soddisfazione.

Il libro di David Cook introduce alla programmazione dei robot, in un viaggio fra resistenze, motori, tensioni, microprocessori.
Si tratta di un libro per molti versi divertente anche se decisamente indicato per un pubblico di fascia alta.

Difficoltà: Alta • David Cook • **Editore:** T-I-A • **ISBN (pbk):** 1-59059-373-1 • **Anno di pubblicazione:** 2004 • **Lingua:** Inglese • **Pagine:** 381 • **Prezzo:** € 34,99